2006-12

Assessing the effects of honeypots on cyber-attackers

Lim, Sze Li Harry

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

ASSESSING THE EFFECT OF HONEYPOTS ON CYBER-ATTACKERS

by

Sze Li Harry Lim

December 2006

Thesis Advisor:	Neil C. Rowe
Second Reader:	John D. Fulp

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** December 2006	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE: Assessing the Effects of Honeypots on Cyber-Attackers		**5. FUNDING NUMBERS**	
6. AUTHOR(S) Sze Li Harry Lim			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000		**8. PERFORMING ORGANIZATION REPORT NUMBER**	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A		**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited		**12b. DISTRIBUTION CODE**	

13. ABSTRACT (maximum 200 words)

A honeypot is a non-production system, design to interact with cyber-attackers to collect intelligence on attack techniques and behaviors. While the security community is reaping fruits of this collection tool, the hacker community is increasingly aware of this technology. In response, they develop anti-honeypot technology to detect and avoid honeypots. Prior to the discovery of newer intelligence collection tools, we need to maintain the relevancy of honeypot. Since the development of anti-honeypot technology indicates the deterrent effect of honeypot, we can capitalize on this deterrent effect to develop fake honeypot. Fake honeypot is real production system with deterring characteristics of honeypot that induces the avoidance behavior of cyber-attackers. Fake honeypots will provide operators with workable production systems under obfuscation of deterring honeypot when deployed in hostile information environment. Deployed in a midst of real honeynets, it will confuse and delay cyber-attackers. To understand the effects of honeypot on cyber-attackers to design fake honeypot, we exposed a tightly secured, self-contained virtual honeypot to the Internet over a period of 28 days. We conclude that it is able to withstand the duration of exposure without compromise. The metrics pertaining to the size of last packet suggested departure of cyber-attackers during reconnaissance.

14. SUBJECT TERMS Fake Honeypots, Deception, Delay, Deterrence			**15. NUMBER OF PAGES** 79
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

ASSESSING THE EFFECTS OF HONEYPOTS ON CYBER-ATTACKERS

Sze Li Harry Lim
Major, Singapore Army
B.Eng., National University of Singapore, 1998
M.Sc., National University of Singapore, 2002

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 2006

Author: Sze Li Harry, Lim

Approved by: Neil C. Rowe, Ph.D.
Thesis Advisor

John D. Fulp
Second Reader

Approved by: Peter J. Denning, Ph.D
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

A honeypot is a non-production system, design to interact with cyber-attackers to collect intelligence on attack techniques and behaviors. While the security community is reaping fruits of this collection tool, the hacker community is increasingly aware of this technology. In response, they develop anti-honeypot technology to detect and avoid honeypots. Prior to the discovery of newer intelligence collection tools, we need to maintain the relevancy of honeypot. Since the development of anti-honeypot technology indicates the deterrent effect of honeypot, we can capitalize on this deterrent effect to develop fake honeypot. Fake honeypot is real production system with deterring characteristics of honeypot that induces the avoidance behavior of cyber-attackers. Fake honeypots will provide operators with workable production systems under obfuscation of deterring honeypot when deployed in hostile information environment. Deployed in a midst of real honeynets, it will confuse and delay cyber-attackers. To understand the effects of honeypot on cyber-attackers to design fake honeypot, we exposed a tightly secured, self-contained virtual honeypot to the Internet over a period of 28 days. We conclude that it is able to withstand the duration of exposure without compromise. The metrics pertaining to the size of last packet suggested departure of cyber-attackers during reconnaissance.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I wish express my gratitude to my thesis advisor, Professor Neil C. Rowe for stimulating my interest in honeypots and JD Fulp for my inaugural lesson in network security.

I appreciate Han Chong, Goh, my fellow “bee” in the honeynet research for the bringing me on deck for this thesis. In addition, I will like to thank Yu Loon, Ng, and Chin Chin, Ng for their timely morale boosting discussion and support.

Lastly, I will like extend my appreciation to my wife, Doris, for her unwavering love and patience, in my absence to complete this thesis.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

> All warfares are based on deception. Hence, when able to attack, we must seem unable; when using our forces, we must seem inactive; when we are near, we must make the enemy believe we are far away; when far away, we must make him believe we are near. Hold out baits to entice the enemy. Feign disorder, and crush him. –
>
> Sun Tzu, the Art of War

The advent of Information Technology (IT) has revolutionized the conduct of commerce, education, socio-politics and military operations. While the technology has been assimilated into our daily lives to increase our competence, capacity and convenience, it has obscured the operational details of these processes. This lack of detailed information offers cyber-attackers an opportunity for exploitations. Information on these exploits is usually restricted to the hacker communities, hence it poses a great challenge for the security community to understand and defend their systems.

The "Honeynet Project" started June 2000 with the mandate to collect intelligence on exploitations and raise awareness of cyber-threats and vulnerabilities. They use and provide tools and techniques, predominantly in the form of honeypots, in their research. Honeypots are systems that are not intended for any production or authorized activity. Hence, any activities, other than those generated by the administrators of the honeypots, are deemed unauthorized or illicit. This value of honeypots lies in the collection of these activities (The Honeynet Project, 2004). Honeynets extend the concept of honeypots to create networks of honeypots, thereby increasing the value that we can derive beyond individual honeypot. However, cyber-attackers soon gain awareness of such intelligence collection tools. This suggests that honeypots could be detected by cyber-attackers, which could reduce the value of honeypots. (Rowe, 2006) suggested to capitalize on the intent of cyber-attackers to avoid honeypots; the concept of fake honeypots was proposed as a defensive technique to deter and delay cyber-attackers.

The objective of this thesis is to learn more through experiments about the decision cycle of cyber-attackers. The goal is to provide appropriate countermeasures by breaking the attackers' decision cycle. We aim to establish the noticeable features of

honeypots from the cyber-attackers' perspective. Capitalizing on the deterrence effect of honeypots, we can design fake honeypots to deter and delay exploitations. Fake honeypots behave like honeypots from the cyber-attackers perspective. They, however, can be production systems and do not need to record exploitation techniques. Due to the fear of detection by the honeypot administrator, coupled with the assumed lack of usefulness of honeypots for normal cyber-attackers, fake honeypots deter cyber-attackers from further reconnaissance and exploitation. A random population of fake honeypots amidst real honeypots in enterprise networks could create uncertainties to confuse and delay the decision of cyber-attackers upon detection of potential honeypots.

The setup of our experiments includes the installation and configuration of a honeypot. We collected and compared results from existing real honeypots and fake honeypots. We analyzed the data to identify evidence of deterrence or "fear" of cyber-attackers. Based on this evidence, we discuss the usefulness of fake honeypots for information-security defense.

The key concepts of the thesis such as honeypots along with the survey of related works, are given in Chapter II. Chapter III gives the problem statement, assumptions and intent. Chapter IV details the test bed setup, configuration, and rationale of the honeypot including hardware, software and network details. Data analysis is given in Chapter IV. Chapter V provides conclusions and suggestions for application. Appendix A includes other results generated from the honeynet. Appendix B includes the source code for the two Java classes.

II. BACKGROUND

This chapter provides background related to our study. The first and second sections provide a process for modeling threats and decision cycles. The third to fifth sections provide the history and overview of honeypots and anti-honeypot/anti-honeynet technology. The sixth section highlight the difference in intrusion detection and prevention systems and their relevancy in the honeypot implementation. The seventh section examines the importance of system integrity in implementation of a honeypot. The last section elaborates on data collection techniques and their associated tools.

A. OBSERVATION, ORIENTATION, DECISION AND ACTION (OODA) LOOP

The key to military victory is to create situations wherein one can make appropriate decisions and translate these decisions into executions more quickly than the adversaries. (Boyd, 1976) hypothesized that all intelligent organisms undergo a continuous cycle of interaction with their environment. Boyd breaks this cycle down to four inter-related and overlapping continuous processes, namely observation (collection of data), orientation (analysis and synthesis of data to form current mental perspective), decision (determination of a course of action based on current mental perspective) and action (physical execution of decisions). This decision cycle is known as the OODA loop.

Analysis of cyber-attacks reveals that the same decision cycle is involved. Penetration and understanding of the cyber-attackers' OODA cycle provides us with a framework to devise an effective security plan.

B. THREAT MODELING IN COMPUTER SECURITY

To devise a cost-effective security plan, we must understand our threat vis-à-vis the value of assets to be protected. Understanding the adversary's view of the system is a critical step in threat modeling process. (Swiderski, 2004) suggested a threat modeling process as shown in Figure 1, which we will follow here.

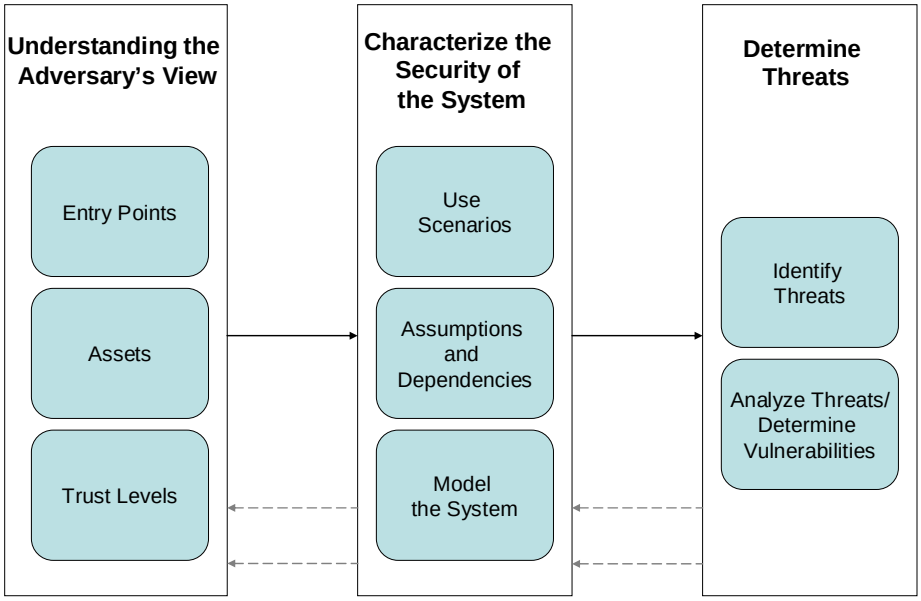

Figure 1. High-level Process of Threat Modeling.

C. HONEYPOTS (THE HONEYNET PROJECT, 2004)

> If you know the enemy and know yourself, you need not fear the result of a hundred battles. If you know yourself but not the enemy, for every victory gained you will also suffer a defeat. If you know neither the enemy nor yourself, you will succumb in every battle.
>
> - Sun Tzu, the Art of War

1. Know Your Enemy

The concept of warfare in cyberspace is similar to that of conventional warfare. Understanding our capabilities and vulnerabilities, vis-à-vis that of the adversaries, allows us to devise defensive and offensive plans. Prior to October 1999, there was very little information about cyber-attacker threats, motives, and techniques. The Honeynet Project was officially incorporated in July 2001 as a nonprofit organization to collect and analyze cyber-attack intelligence to support awareness. Since unique exploit motives and techniques are known only to cyber-attacker communities, and otherwise often not widely known, the Honeynet Project has to devise and employ creative attack-data collection tools like honeypots and honeynets.

2. Definition of Honeypot

"A honeypot is an information system resource whose value lies in unauthorized or illicit use of that resource (The Honeynet Project, 2004)." A honeypot is not designated as a production-oriented component of an information infrastructure. As such, nobody should be using or interacting with honeypots; any transactions or interactions with a honeypot are by definition unauthorized.

3. Variations of Honeypots

There are two categories of honeypots: low-interaction and high-interaction. Low-interaction honeypots are passive and cyber-attackers are limited to emulated services rather than actual operating systems. They are generally easier to deploy and pose minimal risk to the administrators. Examples are Honeyd, Specter and KFSensor. High-interaction honeypots provide entire operating systems and applications for attackers to interact. They are more complex and serve as better intelligence-collection tools. However, they pose higher level of risk to the administrator due to the potential of compromise by cyber-attackers, as for instance, with the use of compromised honeypots to propagate other attacks.

4. Uses of Honeypots

Honeypots can be deployed as production or research systems. When deployed as production systems, different honeypots can serve to prevent, detect and respond to attacks. When deployed as research systems, they serve to collect information on threats for analysis and security enhancement.

5. Honeynets

Similarly to the transition of low-interaction to high interaction honeypots, the value of honeypots can be further extended by networking. Putting honeypots into networks provide cyber-attackers a realistic network of systems to interact with, and permits better analysis of distributed attacks.

6. Virtual Honeynets

Virtual honeynets (The Honeynet Project, 2004) use virtual machines like VMware to emulate multiple systems with different operating systems on a single hardware. While reducing the hardware requirements for the administrators, the virtual guest machines offer cyber-attackers the perspective of independent systems in the networks. It reduces the cost and management for both production and research purposes. There are, however, disadvantages to deployment of virtual honeynets. The use of virtual machines is limited to the hardware virtualization software and the host operating system. The secured management of the host operating system and the virtualization software has to be thoroughly planned and executed. A compromise to either software may allow cyber-attackers to seize control of the entire honeynet. It is easier to fingerprint a virtual honeynet, as opposed to honeynets deployed with real hardware, by the presence of virtualization software and signatures of the virtual hardware emulated by the virtualization software. Cyber-attackers may potentially identify these signatures and avoid these machines, thereby defeating the purpose of deploying the honeynet.

D. ANTI-HONEYPOT TECHNOLOGY

As the security professionals begin to include honeypots or honeynets into their arsenal for information defense, cyber-attackers have reacted with anti-honeypot technology. (Krawetz, 2004) opined that the emergence of these honeypot detection tools suggested that honeypots were indeed affecting the operations of cyber-attackers. This technology can probably be extended to address the detection of diverse honeypots. (Holz and Raynal, 2005) introduced several techniques and tools applicable to cyber-attackers to detect suspicious environments (e.g., virtual machines and presence of debuggers). They presented the detection of Sebek by measuring execution time of the read() system call. Table 1 shows the various implementations of other honeypots and the associated characteristics and potential exploits.

Honeypot/honeynet	Typical Characteristics	Methods for Detecting the Honeypot
BackOfficer Friendly	Restricted emulated services and responses	Send different requests and verify the consistency of responses for different services.
Honeyd	Signature based responses	Send a mixture of legitimate and illegitimate traffic/payload, with common signatures recognized by targeted honeypots.
Symantec Decoy Server or Virtual Honeynet	Virtualization	Detect virtual hardware, for instance Media Access Control (MAC) addresses.
Snort_inline	Modification actions	Send different packets and verify the existence and integrity of response packets.
Virtual Honeynet	System files	Probe for existence of VMware.
Active Tcpdump session or Sebek	Logging processes	Scan for active logging process or increased round-trip time (for instance, due to read(1) in Sebek-based honeypots).

Table 1. Detecting Anti-Honeypot/Honeynet Technology.

E. FAKE HONEYNETS

Capitalizing on the advent of anti-honeypot/honeynet technology, (Rowe, 2006) suggested the use of fake honeypots to deter and delay cyber-attacks. These are real production systems with the signatures or behaviors of honeynets to deter cyber-attacks. The suggested metrics (Rowe, 2006) to guide design of good honeypots may be used by cyber-attackers to detect and avoid potential honeypots. Using the anti-honeypot/honeynet technology during reconnaissance, the cyber-attacker may believe a system with poor metrics score is a honeypot, thus, avoid exploitation of the system. This defensive approach capitalizes on the dislike of cyber-attackers for honeynets. With the current computing and memory capacities, a fake honeynet is most easily implemented on a real machine as a self-contained virtual honeynet.

F. INTRUSION PREVENTION SYSTEM

As part of Defense-in-Depth approach to Information Security, Intrusion Detection Systems (IDS) are commonly deployed to detect potential incoming threats based on signature sets or anomalies. However, such systems are passive; they often overwhelm administrators with alerts instead of timely response to detected attacks. Intrusion Prevention Systems (IPS) are introduced to address this response capability gap.

They extend the detection capability of IDS to include automated controls in response to cyber-attacks, for instance, to block or modify packets (Rash, Orebaugh, Clark, Pinkard and Babbin, 2005). This capability, however, comes at a significant price to the performance of protected networks or systems. Snort is a well-known IDS which will be relevant for post-collection analysis of Tcpdump data in order to detect cyber-attacks against honeypots. Snort_inline builds on the detection capability of Snort. It informs iptables to drop, reject or modify packets in accordance to the rules set. This will be relevant in future investigations to channel cyber-attackers to the path of choice of administrators.

G. SYSTEM INTEGRITY

To implement a self-contained virtual honeynet, we must try to ensure that the host operating system will not be compromised. If it is compromised, we must be able to isolate the compromise. The typical approach to maintaining system integrity is to establish baseline system fingerprints, prior to operations that may result in compromise; for instance, one can connect to Internet and periodically monitor changes to the computer files against these fingerprints. These fingerprints can be cryptographic hashes stored offline to prevent compromise. In case of modifications to these files, the administrator will investigate if the modifications are legitimate and update the offline system fingerprints database accordingly. There many ways to implement such system integrity features. Tripwire, Intact, and Veracity are some high-end commercially available packages. Osiris is an open-source host-integrity monitoring system that keeps the administrator apprised of possible compromises or changes to the file system, resident kernel modules, or user and group lists. Alternatively, scripts or programs can be written to automate the hashing of system files.

H. DATA COLLECTION AND ANALYSIS

(Jones, 2006) categorized the four types of Network-Based Evidence (NBE) for analysis. These include full session data, session data, alert data, and statistical data. Full content data consists of the actual packets, typically including the headers and application information, as seen on the transmission media. This produces complete information but

will take significant disk space. Wireshark (or formerly known as Ethereal) is a memory-intensive application that captures full content data over the network connections, with a good user-interface for full content data analysis. Tcpdump is a lightweight tool used for similar collection. With these tools, analysts are usually overwhelmed with too much information. Richer analysis is normally conducted by examining session data of particularly interesting sessions with tools like Scanmap3d. However, the challenge is often to identify the session of interest. Alert data generated by Snort, shoki and Bro offer intrusion detection alerts through signatures and rules, and these usually represent much less data than the session data. Statistical data on system parameters are often useful for performance monitoring. Such monitoring can trigger new alerts that may not be obvious at lower (more detailed) levels. Tcpstat and Tcpdstat are common open-source statistical tools.

Tcpdump data was the main source of data collected for the project reported here. While Wireshark provided most common processing tools required for the analysis of the Tcpdump data, the processed data did not fit perfectly into our analysis requirement. As such, simple programs had to be developed to extract and cluster the data.

THIS PAGE INTENTIONALLY LEFT BLANK

III. PROBLEM DEFINITION AND ASSUMPTIONS

A. PROBLEM DEFINITION

Without constant innovation and improvement in the deployment of honeypots, their effectiveness will decrease with the increasing awareness of the cyber-attackers. Signatures to detect honeypots could be available to enable cyber-attackers to avoid honeypots much as they are for permitting defenders to recognize attacks. Till the advent of newer cyber-attack intelligence collection tools, we need to maintain the effectiveness and relevancy of honeypots. While cyber-attackers have devise mechanisms to evade our detection and protection system, we can, likewise, deploy real and fake honeynets to create confusion and obfuscation evade their reconnaissance and attacks.

Honeypots have been deployed to prevent, detect and respond to attacks, however, they are usually restricted to conventional passive deployment at enterprise level. The deployment of an active self-contained virtual honeynet as a preventive security measure for the individual host terminal, rather than as a data collection tool, has not been tested. This thesis explores the use of a self-contained virtual honeynet as part of defense-in-depth mechanisms for a rapid response deployment scenario where operators are constrained by the small logistics footprint.

B. ASSUMPTIONS

1. Threat Model

For purpose of this thesis, we categorize cyber-attackers into the following groups.

a. Ignorant Cyber-Attackers

These cyber-attackers are assumed to be ignorant of the existence of honeypot technology and do not understand its nature. They consist of “script kiddies” or novice amateurs with little experiences in offensive information operations. Their wares are often downloaded from the Internet or distributed by fellow cyber-attackers. They are motivated by thrill, challenge, pleasure, recognition and occasionally profit (in terms of “owning” machines as bots). Most of them can be prevented by standard good security management of systems.

b. Honeypot-Aware Cyber-Attackers

These are aware of honeypot technology and use a combination of automated hacking tools, armed with signatures of potential honeypots, and manual attack techniques. They may be experienced in offensive information operations. They use risk assessment and are cautious in their navigations and attacks. Generally they will be "scared" off by the existence of honeypots since there are so many easier targets. They may be motivated by knowledge and mission requirements. This is the ideal group target for fake honeypots.

c. Advanced Cyber-Attackers

These have detailed knowledge of honeypots. They are usually goal-specific in nature, able to stealthily probe potential honeypots to distinguish real and fake ones. Fake honeypots are not designed to deceive this group. However, when deployed amidst real honeypots, it is expected that fakes will obscure the real ones, hence slowing the decision cycle of these cyber-attackers. In addition, it may create frustration for these cyber-attackers, leading to emotional, rather than logical, reactions (Rowe, 2006).

C. GOAL

The goal of this thesis is to establish the existence of different behavior of cyber-attackers towards the honeypots. Subsequently these reactions can be exploited in fake honeypots to deter honeypot-aware cyber-attackers. We will also explore the feasibility of using honeypots to delay advanced cyber-attackers through obfuscation and confusion.

IV. EXPERIMENTAL SETUP

This chapter details the hardware, software, and layout of the honeynet used in experiments. We report the problems encountered in setting up the experiment. We also discuss the motivation behind the Java program to analyze Tcpdump data.

A. EXPERIMENT SPECIFICATION

1. Hardware Specification

The hardware setup prepared by (Duong, 2006) in her M.S. thesis was maintained as a control setup of a real honeynet. It consisted of three computers, namely the Router, Honeynet and Data Capture machine. In addition, a Dell Inspiron Laptop was set up with a self-contained virtual honeynet to serve as the fake honeynet. Instead of a full-fledged router machine, a Belkin Router was used to perform create the network address translation (NAT). Table 2 and Figure 2 lists and illustrates the specification,, respectively.

Router (Dell Dimension XPS B933)	
Processor	Intel Pentium III – 933MHz
Storage	Maxtor (Ultra ATA) – 20GB
Memory	512MB
NIC	Davicom Semiconductor 21x4x DEC Tulip – Compatible 10/100Mbps
	3Com 3C905C – TX Fast Etherlink 10/100Mbps
	3Com 3C905C – TX Fast Etherlink 10/100Mbps
Drives	DVD-ROM, CD-RW, Zip, Floppy
Honeynet (Dell Optiplex GX520)	
Processor	Intel Pentium 4 -2.80GHz
Storage	(Serial ATA) – 40GB
Memory	1024 MB
NIC	DELL NetXtreme BCM5751 Gigabit Ethernet PCI Express (integrated)
Drives	CD-RW, Floppy
Data Capture (Gateway)	
Processor	Intel Pentium 4 – 1.80GHz
Storage	Western Digital (Ultra ATA) – 40GB
Memory	256 MB
NIC	EthernExpress Pro/100 VE (integrated)
Drives	CD-RW, Floppy
Fake Self-contained Honeynet (Dell Inspiron 6000)	
Processor	Intel Pentium 4 – 1.86GHz
Storage	Seagate (Ultra ATA) – 80GB
Memory	1.25 GB
NIC	Broadcom 440x 10/100 Integrated Controller
	Dell_Wireless WLAN Card
Drives	NEC ND-6650A 8x DVD+/-RW
Router	Belkin Wireless G Router

Table 2. Experimental Hardware Specification.

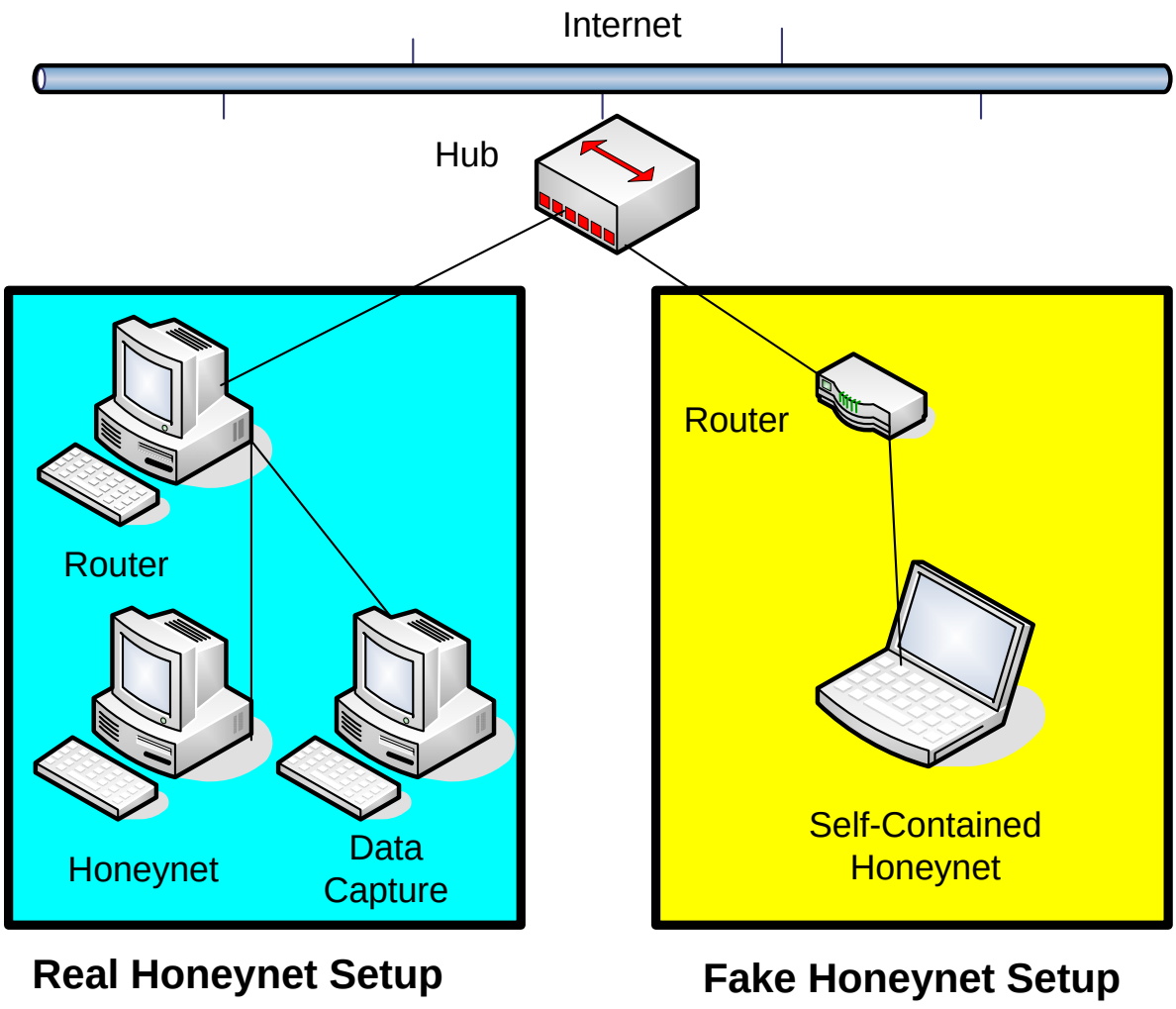

Figure 2. Hardware Setup.

2. Software Specification

The software specification from Duong's honeynet setup was maintained. In addition, SUSE Linux 10 was installed on the Fake Honeynet machine. The intent was to replicate the software configuration of the honeynet to provide common basis of comparison.

In light of the increasing popularity of scanners for VMware as means to detect honeynets, VMware 5.5 was setup to host a Microsoft Windows XP Professional with Service Pack 2 and a Microsoft Windows 2000 Advanced Server with Service Pack 4. Figure 3 shows the fake honeynet setup.

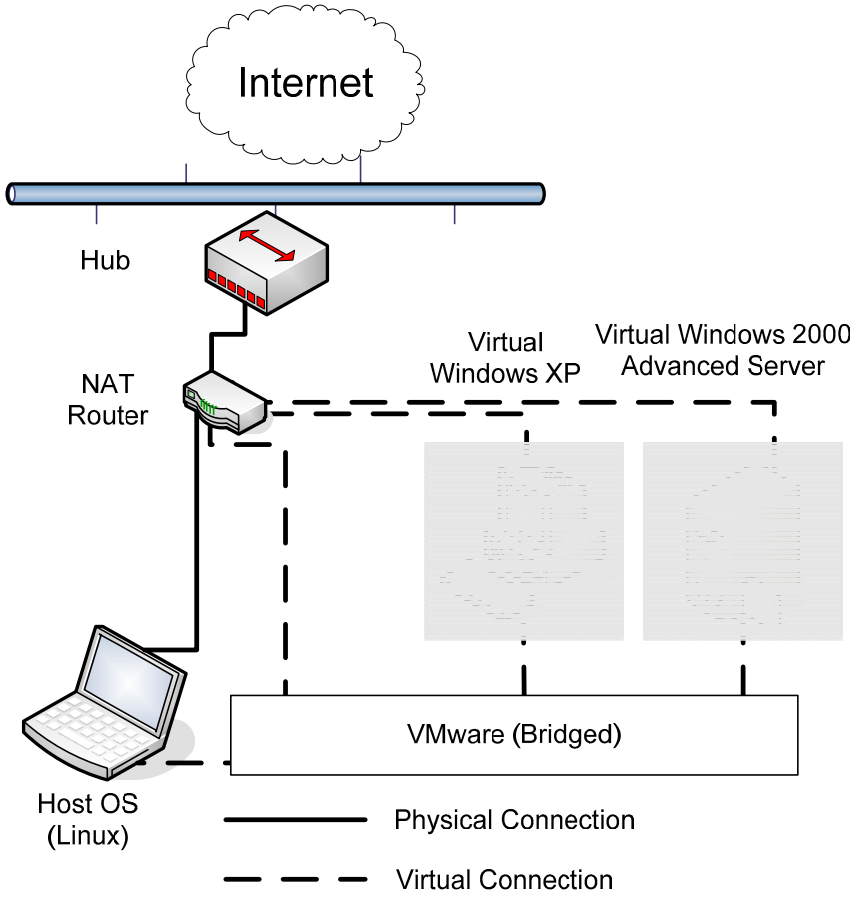

Figure 3. Fake Honeynet Setup.

As the two guest operating systems were to simulate honeypots, they were not fully equipped with all common applications and services. As it would be uncommon for a user to use machines without common applications like Microsoft Office, this served as another attempt to fool attackers as to the presence of a honeynet. Lastly, the Snort intrusion-detection system and Tcpdump packet collector were installed, though only Tcpdump was running as an active process. Tcpdump was selected due to its lightweight ability to capture full content data as opposed to other tools. This served to create the impression of active logging characteristic of honeynet. All software was fully patched with the latest updates, and the baseline fingerprints for the host Linux machines were obtained using the Host System Integrity Monitoring System Osiris 4.2.2 prior to connection to Internet. As opposed to the real honeypot, the integrity of the guest operating systems (OS) on the fake honeynet, namely the Windows XP Professional and

Windows 2000 Advanced Server, is of lesser concern as they are not part of the production systems. The only concern is that that they may be "owned" or compromised and used as launch pad for other attacks. This risk can be significantly reduced by frequent remounting of fresh guest operating system and activation of automatic updates.

The analysis machines were not included in the experiment setup. They were used to test for network connectivity and perform troubleshooting. Wireshark was installed on a separate machine to facilitate analysis of full content data.

Router (SUSE Linux 10)	
Primary Goal	Sniff traffic, send captured data to Data Capture
Software	Snort 2.4.3 – intrusion detection system
	Tcpdump – packets capture
Honeynet (SUSE Linux 10)	
Primary Goal	Solicit attacks
Storage	Tcpdump – packets capture
	VMware Workstation 5.5 hosting
	Windows 2000 Advanced Server with SP4
	Windows XP Professional with SP2
Data Capture (SUSE Linux 10)	
Primary Goal	Store Snort Data
Storage	PostgreSQL 8.1.1
Fake Self-contained Honeynet (SUSE Linux 10)	
Primary Goal	Solicit/Deter attacks
Storage	Osiris/Unix integrity checker
	Tcpdump – packets capture
	VMware Workstation 5.5 hosting
	Windows 2000 Advanced Server with SP4
	Windows XP Professional with SP2

Table 3. Experiment Software Specification.

B. DESIGN OF THE EXPERIMENT

Full-content Tcpdump data was collected at the Ethernet interface over a period of 28 days for the fake honeynet. The Pcap dump file was collected on a weekly basis for analysis. Similar data was conducted for the real honeynet over a period of 21 days.

Wireshark was used to process the Pcap file into readable text. We were, in particular, interested in the TCP sessions. They can be obtained using the Statistics|Sessions commands. To facilitate subsequent analyses, the name resolution box was unchecked to allow the unresolved addresses to be displayed instead of default resolved addresses.

In addition to the session information, we would like to find similar information between a pair of Internet Protocol (IP) addresses. We termed this as IP connection. The last packet of interest, however, -- a particular interest of ours -- was not offered by standard built-in utility of Wireshark. Selected displayed fields from Wireshark were captured for further analysis. They included:

Field	Packet Description
Serial Number	Serial number of the packet
Time	Time elapsed since issuance of a collection command.
SrcIP	Source Internet Protocol address
SrcPort	Source port
DestIP	Destination Internet Protocol address
DestPort	Destination port
Size	Size of packet in bytes

Table 4. Field Values of Packets Obtained from Wireshark.

Two simple Java classes were developed to digest these exported files. The first "tcpdumpAnalysisConnectionLastPacket" read the exported files from Wireshark,

filtered the last packet of each connection between two machines (based on their IP addresses), and generated an output file containing the meta-data of the last packet of each connection. The second "tcpdumpAnalysisConnectionSocketPairSizeCountTime" class read the IP address of interest and the exported files from Wireshark, and generated the cumulative size and count of each socket-pair across time. It provided an output file containing the meta-data of the last packet of socket pair of interest. The procedure followed by these programs is illustrated in Figures 4 and 5.

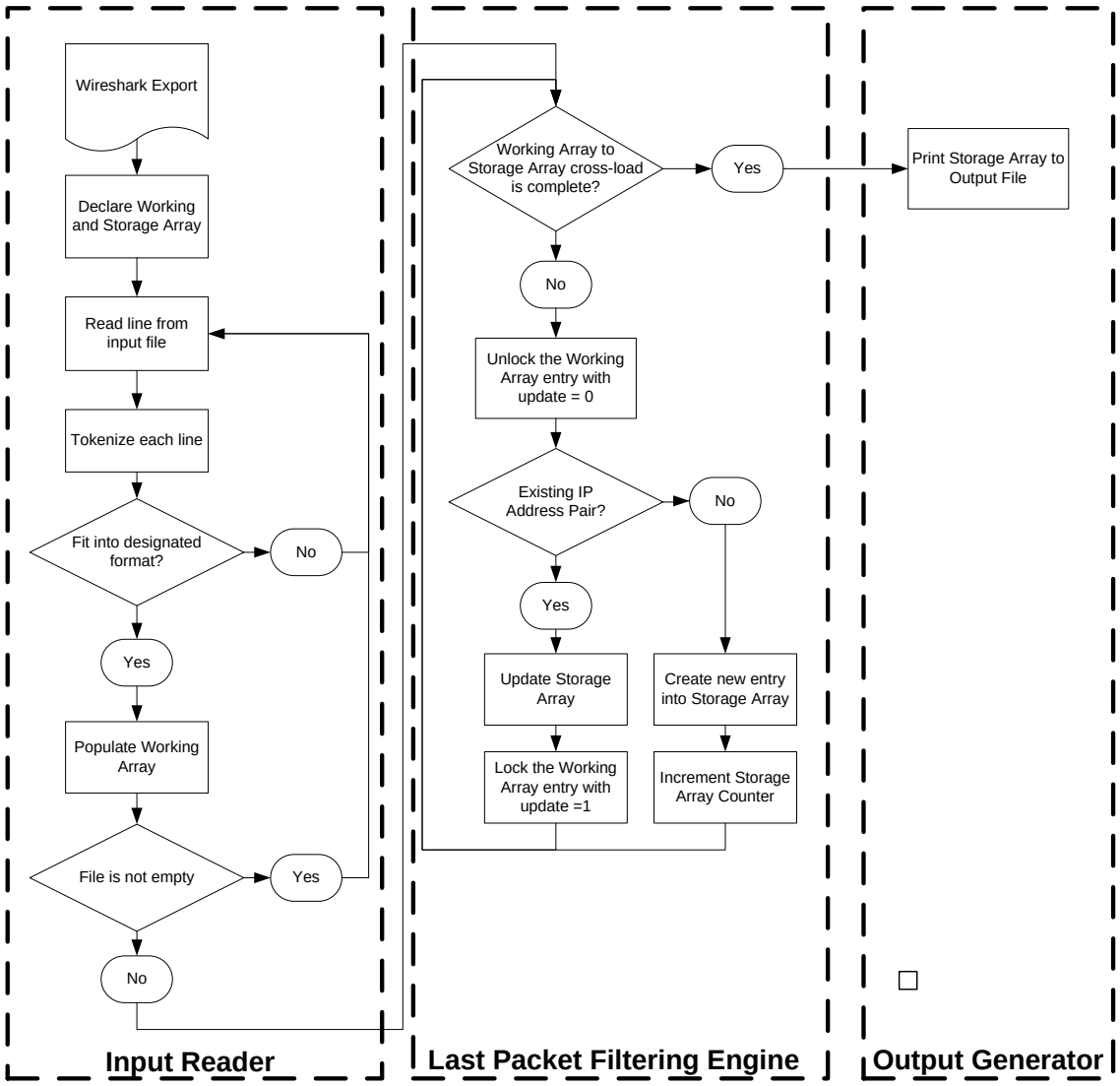

Figure 4. Algorithm of tcpdumpAnalysisConnectionLastPacket class.

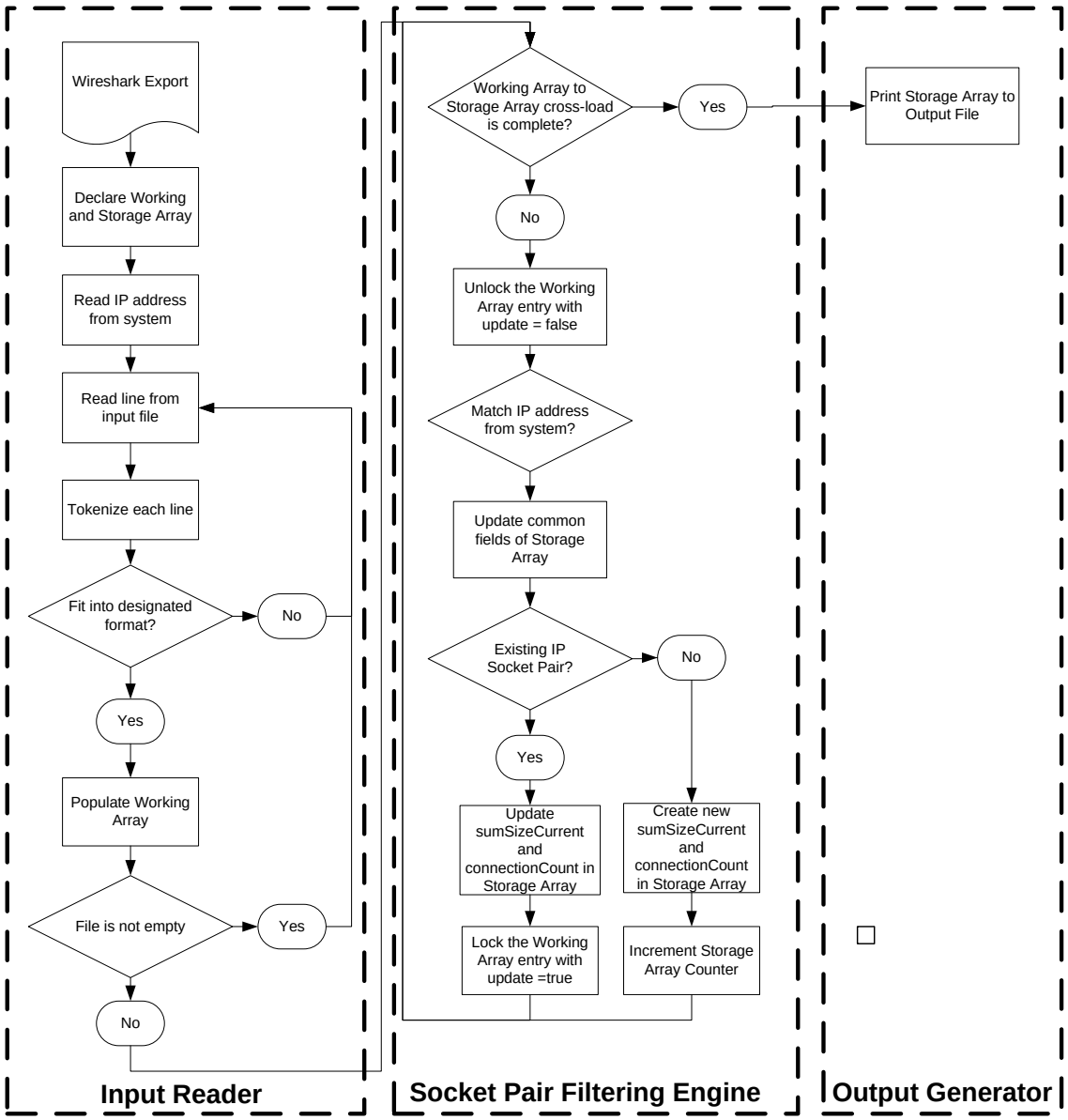

Figure 5. Algorithm of tcpdumpAnalysisConnectionSocketPairSizeCountTime class

V. ANALYSIS OF RESULTS

This chapter details the analysis of results of the collection of the Tcpdump data. It highlights network traffic trends observed which may be useful for future investigation.

A. SECURITY OF FAKE HONEYNET

One goal of this investigation was to determine the effectiveness of the fake honeynet to deter or confuse the cyber-attackers. The intent was not to employ it as the core security mechanism but as part of a comprehensive suite of secured management for information systems to achieve defense-in-depth. Using standard practices for secured management of information systems, the fake honeynet was patched and tightened. The initial system fingerprints were generated using an MD5sum script and Osiris was installed to manage the system integrity. The guest and host operating systems interfaced with the Internet behind a Belkin router that provided NAT. Standard application firewalls were, in addition, maintained to prevent hostile traffic from reaching the host operating system. We ran the collected Tcpdump data through Snort 2.4.5 and received no alerts that suggested successful break-in to host and guest operating system. All alerts received are of Priority 3 (low priority). In addition, our scanning of our network-based evidence showed no break-in behavior, as for instance the installation of rootkits. Along with the results of our system integrity check, we can assert with some confidence that the host operating system was not compromised in the 28 days of exposure.

B. TRAFFIC VOLUME OF HONEYNET

Due to the limitations of our experimental design, the good state of security of the host operating system offered limited insights into the decision cycle of the cyber-attackers. Figure 6 shows the number of TCP sessions for the fake and real honeynets.

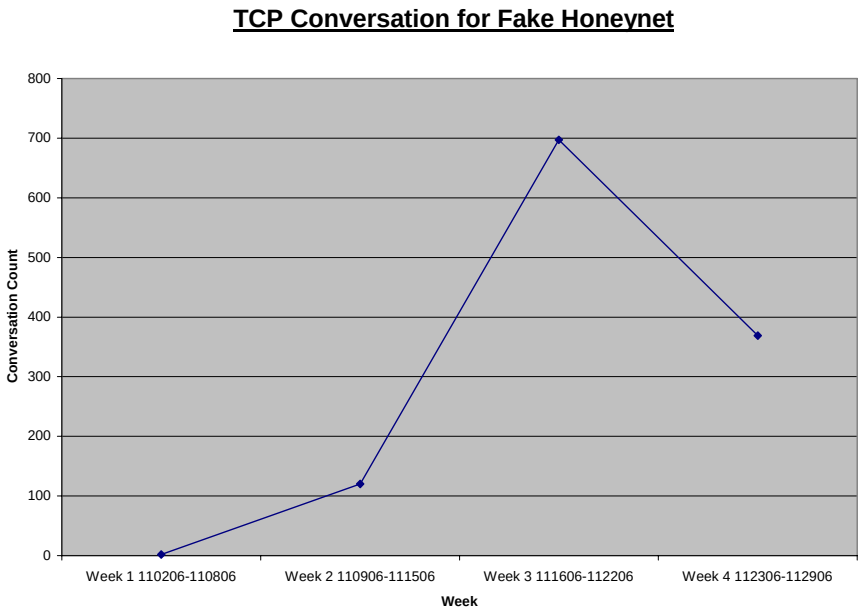

Figure 6. Plot of TCP Session Count for Fake Honeypot Across Weeks.

The traffic on the fake honeynet was significantly lower than traffic of the real honeynet. For the first week, the fake honeynet was set up at home with the Internet Service Provider (ISP) of Comcast. We attributed this to the inaccessibility of the IP address. In the second week, we shifted the fake honeynet to the same subnet of the real honeynet and TCP traffic volume increased significantly. The IP address of the real honeynet had been available to hackers for almost a year and had been disseminated online. As such, its subnet could be deemed a hot zone for TCP traffic.

In the third week, we experimented with the advertisement of both the real and fake honeynets in Web logs (blogs) and hackers' discussion forum. The effect of advertisement was a 6-fold increase in TCP traffic. In addition, we received information, from the hackers' discussion forum, pertaining to our system configurations and our physical address. There were comments that our fake honeynet was so tightly maintained that it was difficult for an amateur hacker to break into our system. This further supported our assertion on the security of the fake honeynet. In the fourth week we observed a 50% drop in our TCP traffic, possibly due to the loss of interest in the honeynet. The results of their reconnaissance might have suggested that they were not adequately equipped with tools for further reconnaissance or exploitation. In addition,

the fourth week coincided with the Thanksgiving weekend, which may have affected the American hackers, though the wide distribution of attack times suggested that we were also getting attacks from all over the world. Either explanation supports the hypothesis that most cyber-attackers that were visiting our honeynet were amateur. While professional awareness of anti-honeypot technology began in 2003, it appeared that the amateurs visiting our honeynet still lacked awareness or tools pertaining to anti-honeypot technology. So it appeared that the fear or deterrent effect could not be observed with the existing amateur cyber-attackers.

C. BELIEVABILTY OF FAKE HONEYNET

To deceive, we needed to establish believability of a honeynet from the perspective of the cyber-attackers. We attempted to duplicate the setup of the real honeynet within the constraints. The lightweight self-contained constraint helped deployability while maintaining a high level of interaction. The fully functional guest and host operating systems were connected to the Internet through a bridged configuration, where the VMware emulated multiple virtual network interfaces with the single physical network interface of the host system. The Belkin router G was used to provide Dynamic Host Configuration Protocol (DHCP) services and assigned each operating system a unique IP address. This would be a typical network configuration for home or office, especially with the increasing popularity of wireless home or office networks. From the perspective of the cyber-attackers, it was difficult to decide if the downstream machines were physical or virtual machines, let alone real or fake honeypots.

Figure 7 compares the number of TCP sessions for real and fake honeynet. The figures for each week varied significantly. Reference the discussion above, the variation was attributed to the established IP address used by the real honeynet. Ignoring week 1, it was interesting to note that session counts were increasing and decreasing similarly over the subsequent weeks. Hence we could conclude that the fake honeynet were not significantly different from the real honeynet from the perspective of the cyber-attackers over the network.

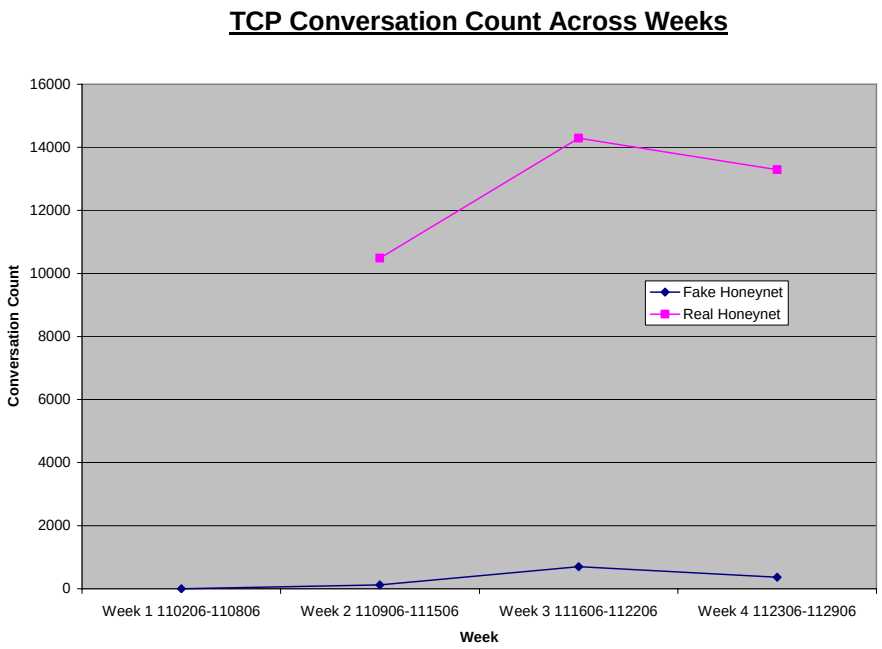

Figure 7. Plot of TCP Session Counts Across Weeks.

D. SESSION ANALYSIS

The session statistics were generated by Wireshark. The ratio of received to sent bytes for each honeynet machine was plotted against the session number with sessions numbered in order of occurrence, with the associated IP address listed for each session. The plots for Windows 2000 Advanced Server were used for discussion in this chapter, and the others are in Appendix A.

Figures 8, 9, and 10 show plots for weeks 2, 3, and 4. In week 2 the ratio value ranged from 0.76 to 12.93. Based on similar observations, we concluded that ratios above three suggested legitimate sessions between the two machines, for instance, a download of data from the designated IP port address. Ratio values within the limits of one to three indicated session support activities like an attempt to set up a session or request services. Excluding the first data point, Figure 9 illustrates session support activities where the ratio values vary from 1.68 to 2.93, where the machine was performing a domain name system (DNS) query to our DNS server. The "recursion desired" flag was set to one and this resulted in many queries to that IP port address. Further inspection of the associated raw packets revealed that this behavior was triggered

by Windows 2000 Advanced Server in our honeynet with intended destination of update.microsoft.com. We assumed this to be benign traffic, but a similar traffic profile involving a different remote site might have been deemed a port scan and triggered alerts. This suggested the need to be turn off automatic updates when setting up honeypot/honeynet to remove these alerts, as it might mask other interesting traffic from cyber-attackers. The administrator, however, can schedule manual updates in a controlled fashion, in order to maintain the currency of the system. In fact, other than initial network diagnostics to confirm network connectivity, a honeynet should refrain from usage like web-browsing and downloading of application or updates, to prevent the generation of unnecessary traffic. The spike in Figure 10 is the download of updates from Microsoft Corporation whereas the ensuing traffic could be a suspicious port scan.

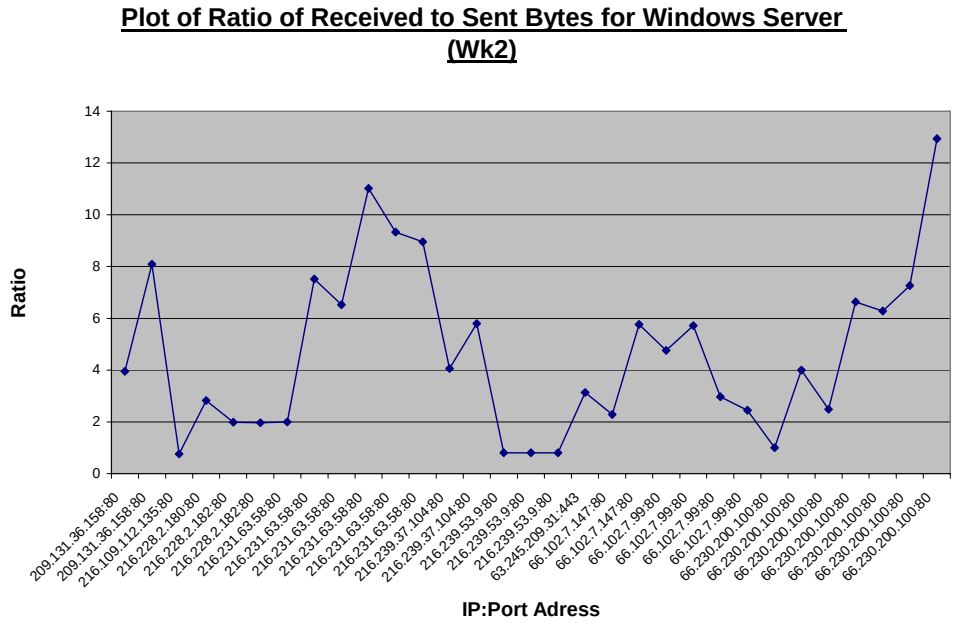

Figure 8. Plot of Ratio of Received to Sent Bytes for Windows Server (Week 2).

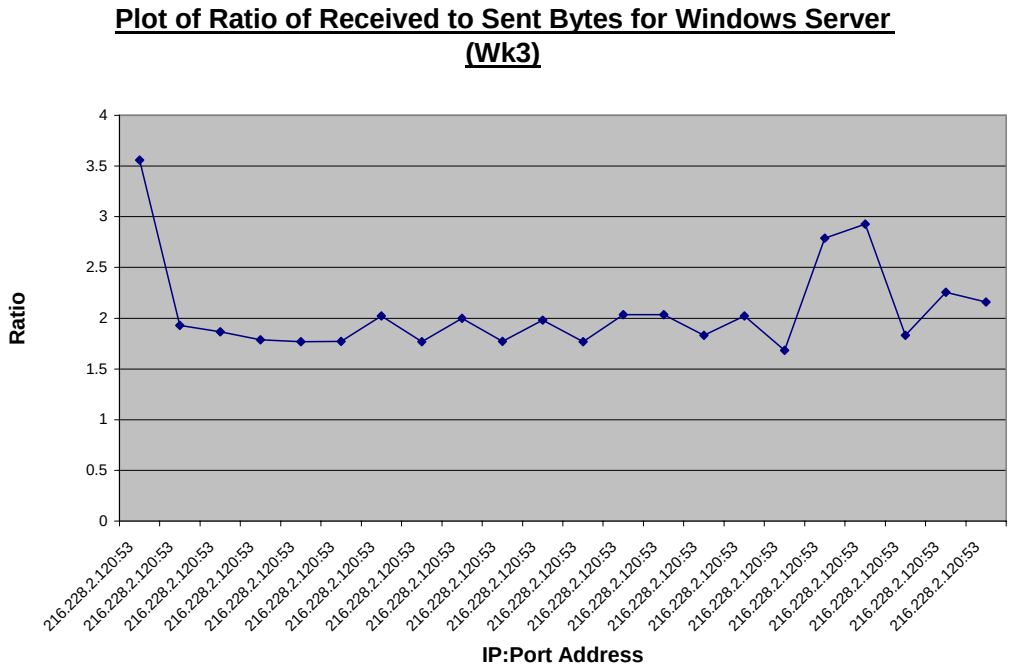

Figure 9. Plot of Ratio of Received to Sent Bytes for Windows Server (Week 3).

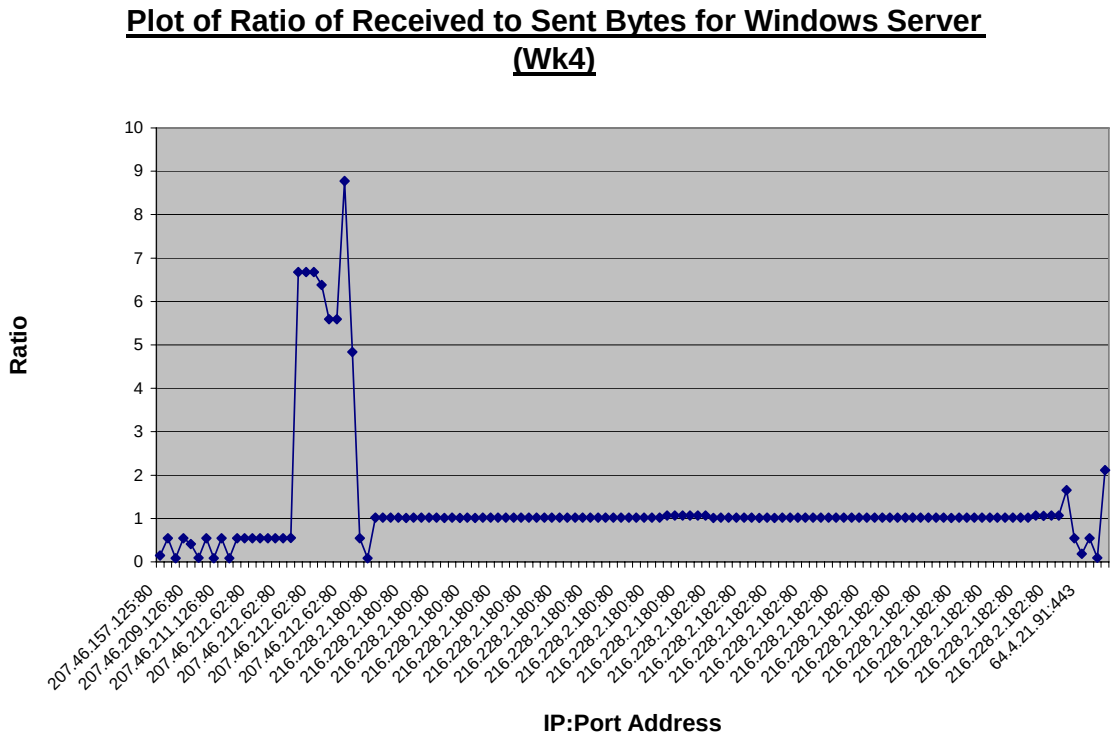

Figure 10. Plot of Ratio of Received to Sent Bytes for Windows Server (Week 4).

E. TIME DOMAIN ANALYSIS

We coded a simple Java program to add two fields to the each packet obtained from the Tcpdump data. The fields were cumulative count and cumulative size of the packet for each session. We further processed the data to obtain the ratio of actual cumulative size to estimated cumulative size, where the estimated cumulative size is the product of current packet size and quantity of packets received in the session. Under normal circumstances, the ratio should be close to one to indicate a consistent flow of bytes between the two sockets. It was however not predictably observed from our data.

A trend of the ratio nearly constant might indicate a standard automated scan, for instance a TCP SYN flag scan, or it could be a perfectly legitimate session with packets of similar sizes. However, a legitimate session should usually terminate with a spike for the ratio. This assumes that the best way to transfer bytes across the network should maximize the bandwidth to the limit of the maximum transfer unit (MTU), as for instance 1500 bytes for the Ethernet. As the last packet transferred under such circumstances might not be a full 1500 bytes, this could result in a spike in the ratio.

Figure 11 shows that the host Linux operating system had high-intensity packet traffic at the beginning and at 360000 seconds. In addition, there were also four clusters of relatively low-intensity traffic. We realized that the four clusters corresponded to daily system network maintenance, whereas the first spike indicated the traffic created during the initial honeynet setup. Since the honeynet setup was performed on a Thursday, the second spike corresponded the service check conducted on the following Monday.

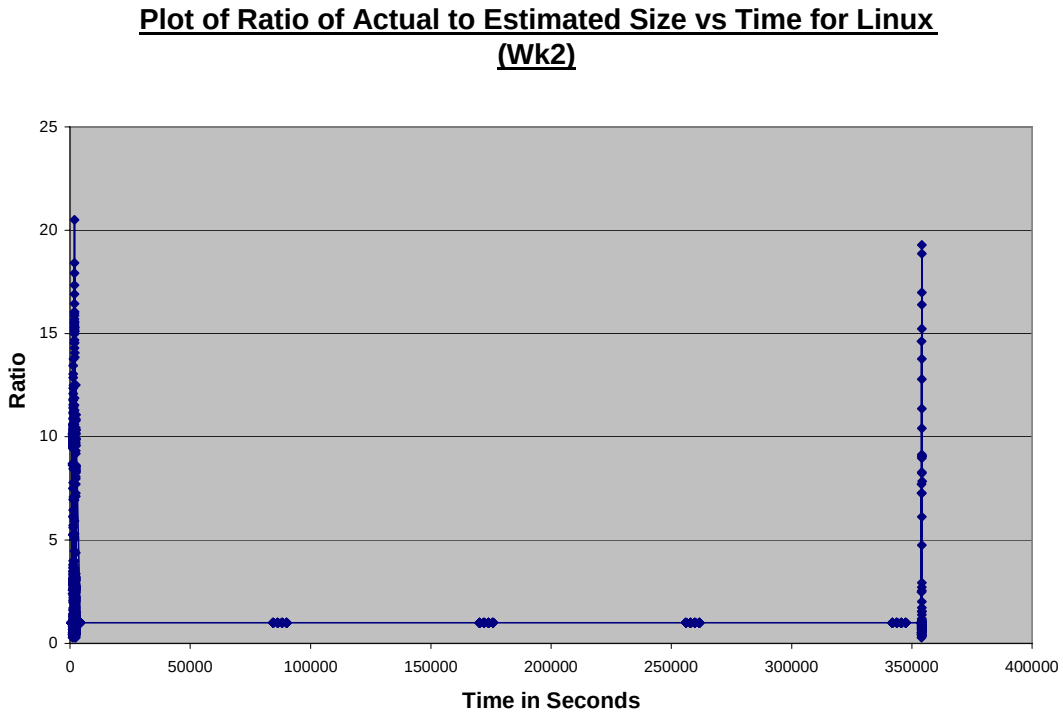

Figure 11. Plot of Ratio of Actual to Estimated Size against Time for Linux Host Operating System (Week 2).

Figure 12 showing the traffic on the Windows Advanced Server is considerably more eventful. It illustrates the effect of our IP address advertisement on blogs and hacker discussion forums. We observed high intensity traffic in the around the ratio of 1. In addition, the relatively high intensity of data points in the vertical direction indicated that packet traffic occurred very quickly. Since this was a honeynet machine, traffic of such extent and intensity should not be observed unless there were automated reconnaissances. The second observation was the relatively consistent ceiling at a ratio of 24. Assuming that software made the best effort to maximize of the utility of the 1500 bytes MTU, we could see there were numerous interweaving scanning packets of approximately 60 bytes with other potentially legitimate data transfers (at 1500 bytes).

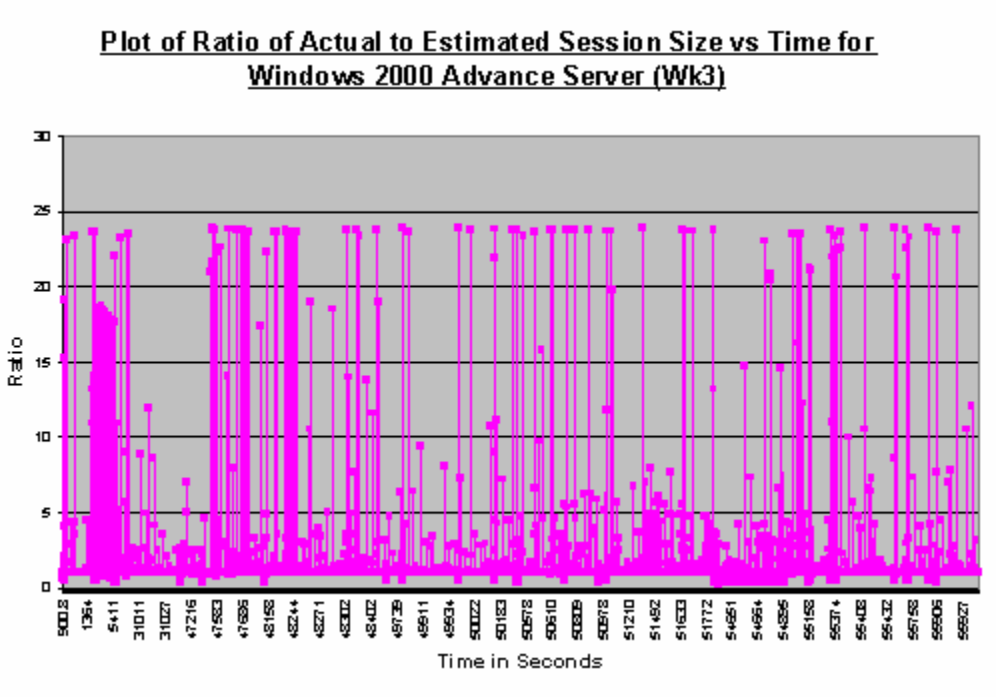

Figure 12. Plot of Ratio of Actual to Estimated Size against Time for Windows 2000 Advanced Server Operating System (Week 3).

F. PACKET SIZE ANALYSIS

We attempted to observe the trends in the distribution of size of packets received by the honeynet. Figures 13, 14 and 15 shows the frequency of size of each packet received by the honeynet collected over three weeks,, respectively. The modes of week 2 and 3 indicate high frequency of data transfer where the packet sizes reach the limit of the Ethernet MTU. This is commonly observed in any Ethernet network. The mode for week 4 indicates that a relatively high frequency of packet sizes between 50 to 100 bytes. This may be useful to indicate reconnaissance activities. We investigated further into these small packets in the next section.

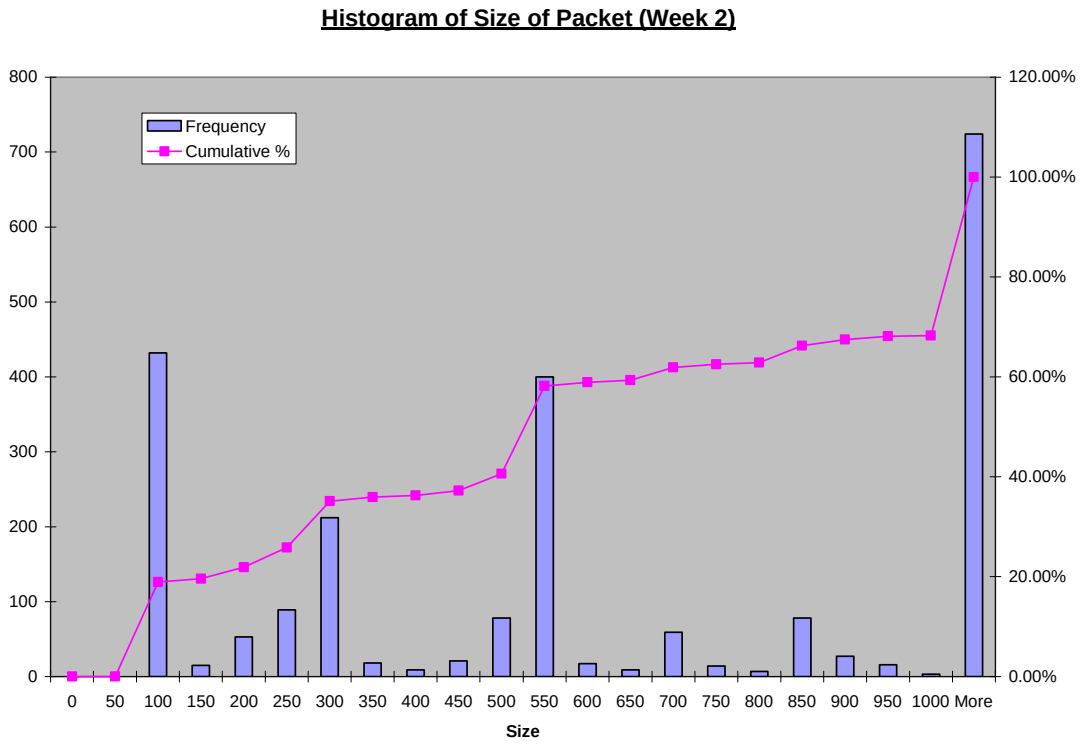

Figure 13. Histogram of Size of Packets Received by Fake Honeynet (Week 2).

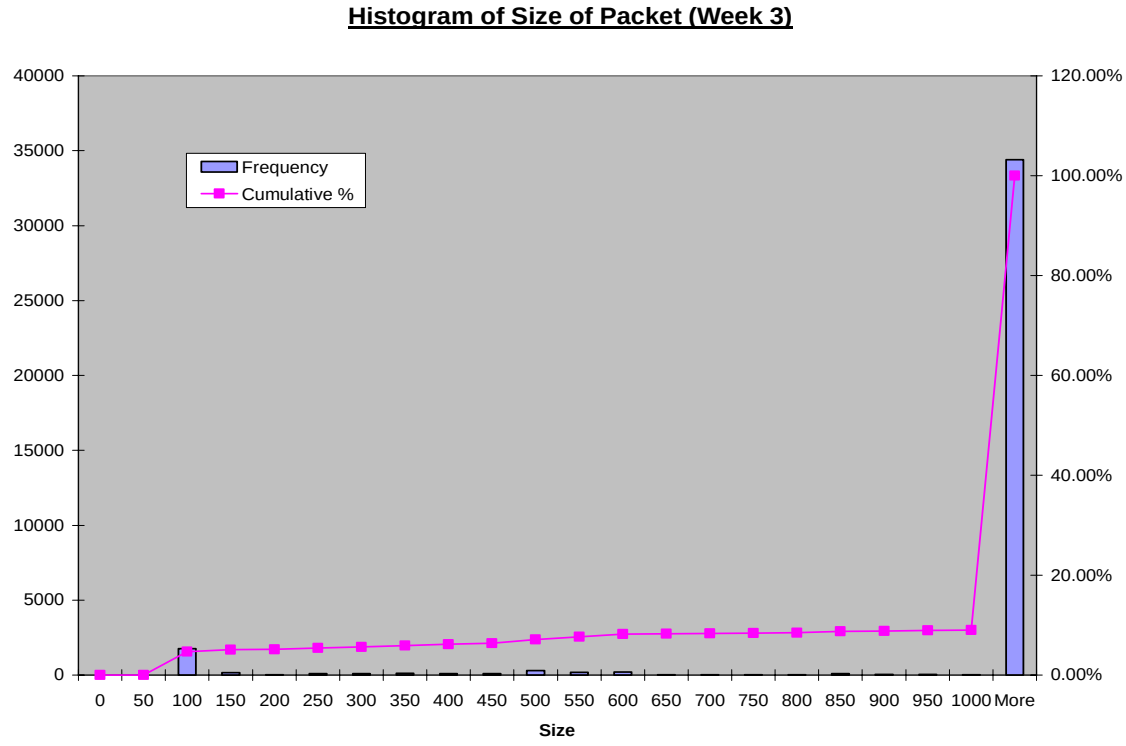

Figure 14. Histogram of Size of Packets Received by Fake Honeynet (Week 3).

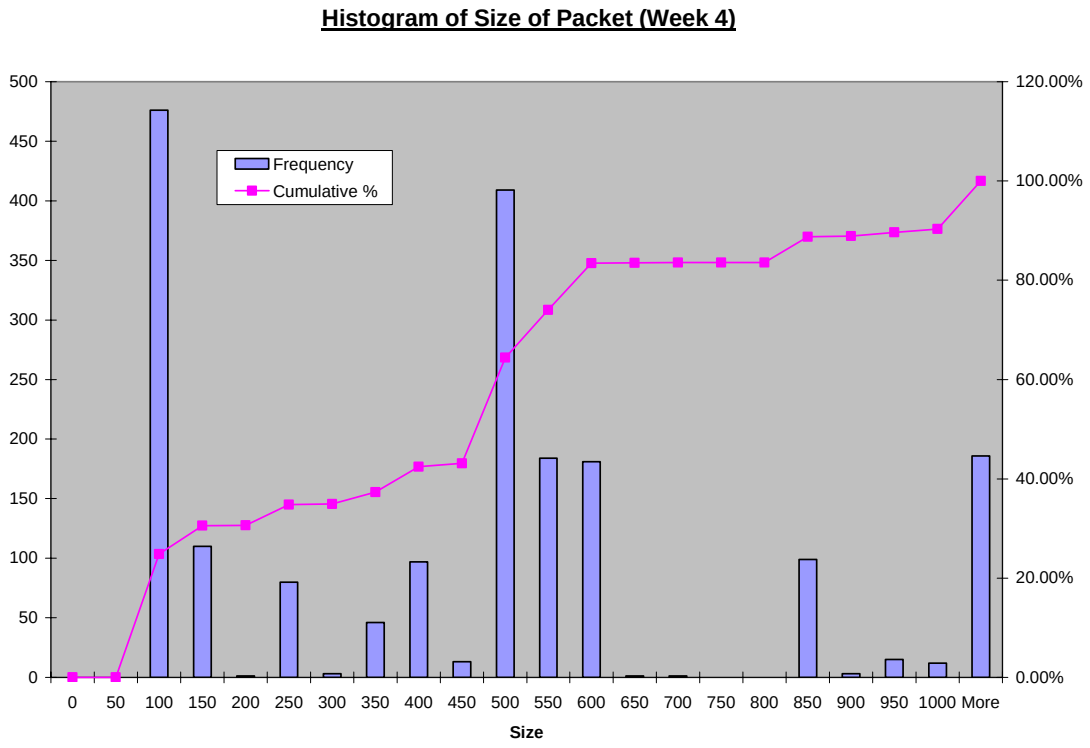

Figure 15. Histogram of Size of Packets Received by Fake Honeynet (Week 4).

G. LAST PACKET RECEIVED ANALYSIS

We developed another Java program to extract information pertaining to the last received packet of each connection between two IP addresses (regardless of ports). A connection between two machines could be made up of several sessions. Since the honeynet was not designed to be a production system, termination of a connection might indicate a loss of interest or actual fear of the honeynet. The results generated from the Tcpdump data using the Java class was plotted into histograms as shown in Figures 16, 17 and 18. It can be observed that these plots are cleaner modulated plots of Figures 13, 14 and 15, respectively. The histograms revealed that three distinct regions of sizes with relatively high frequency. They were last-packet sizes of 50-100, 500-600, and 900-950 bytes. Following the above discussion on small packets, we tabulated the percentage of traffic where the last-packet size was between 50 and 100 bytes, in Table 5.

Data Set	Percentage of Traffic
Week 2	44%
Week 3	79%
Week 4	10%

Table 5. Percentage of Traffic with Packet Size Between 50 and 100 Bytes.

Week 3 was the week when we solicited traffic through active advertisement. Near to 80% of the connections ended with packets of size between 50 to 100 bytes. While we could not assert if the cyber-attackers were aware of the existence of our fake honeynet, the above traffic might fit into a typical departure signature during reconnaissance.

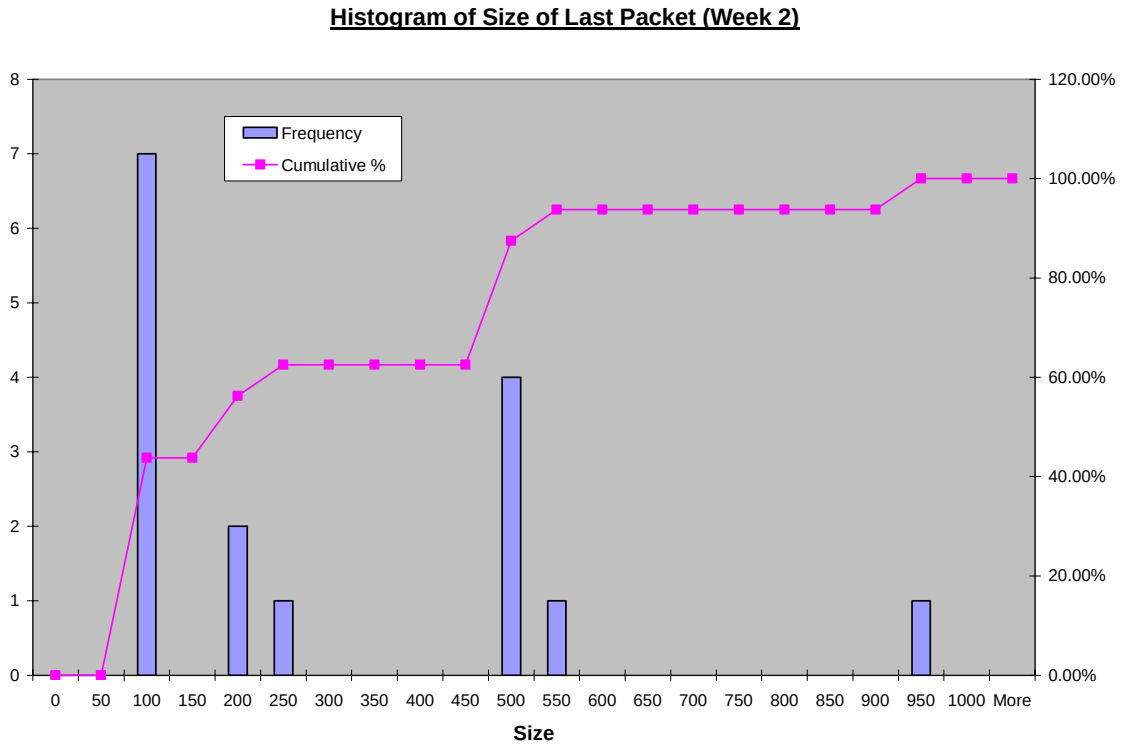

Figure 16. Histogram of Size of Last Received Packet by Fake Honeynet (Week 2).

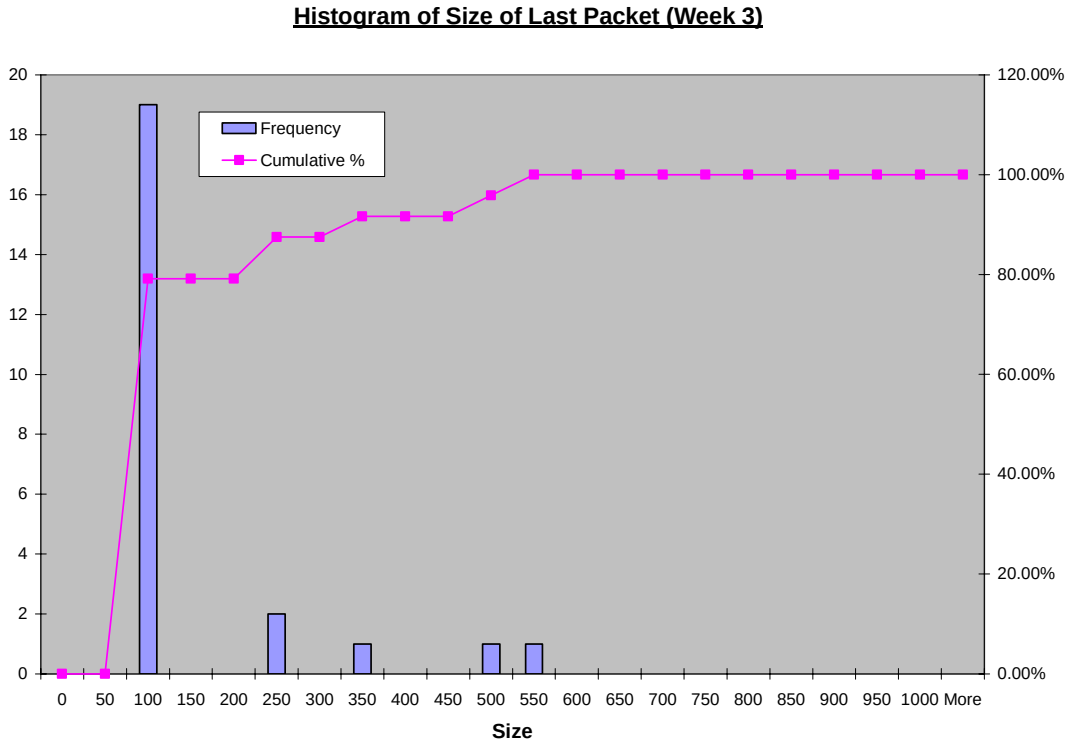

Figure 17. Histogram of Size of Last Received Packet by Fake Honeynets (Week 3).

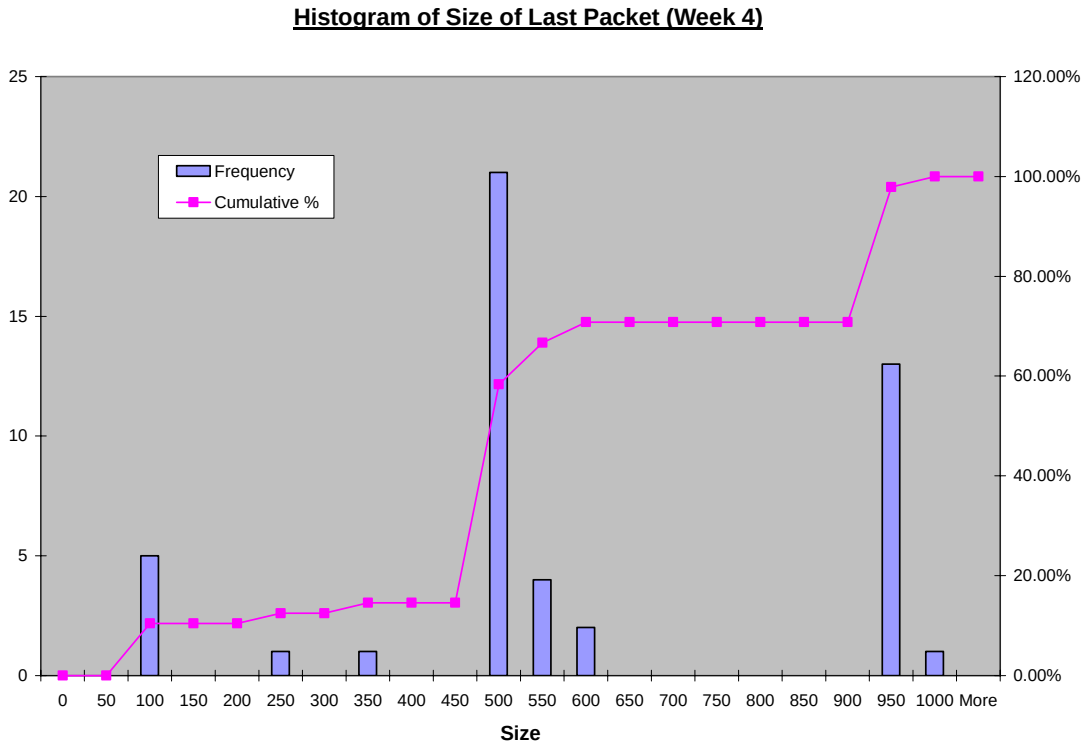

Figure 18. Histogram of Size of Last Received Packet by Fake Honeynets (Week 4).

We further investigated into the distribution of protocol and flags of the last received packets for our self-contained virtual honeypots. Tables 6, 7 and 8 shows the distribution of the protocol and flags set for the last packets received by our self-contained virtual honeypots .

Source IP	Protocol	TCP	UDP	ACK	PSH	RST	SYN	FIN	DON'T FRAG	MORE FRAG
221.208.208.92	Messenger	0	1	0	0	0	0	0	0	0
62.213.130.127	Messenger	0	1	0	0	0	0	0	0	0
221.208.208.83	Messenger	0	1	0	0	0	0	0	0	0
202.97.238.196	Messenger	0	1	0	0	0	0	0	0	0
202.97.238.203	Messenger	0	1	0	0	0	0	0	0	0
216.239.53.9	TCP	1	0	1	0	1	0	0	0	0
216.239.37.104	TCP	1	0	1	0	0	0	0	0	0
216.231.63.58	TCP	1	0	1	0	0	0	0	0	0
66.102.7.99	TCP	1	0	1	0	0	0	0	0	0
66.230.200.100	TCP	1	0	1	0	0	0	0	0	0
216.228.2.120	DNS	0	1	0	0	0	0	0	0	0
66.102.7.99	TCP	1	0	0	0	1	0	0	0	0
216.228.2.120	DNS	0	1	0	0	0	0	0	0	0
66.102.7.99	TCP	1	0	0	0	1	0	0	0	0
216.228.2.120	DNS	0	1	0	0	0	0	0	0	0
216.228.2.120	DNS	0	1	0	0	0	0	0	0	0
Total		7	9	5	0	3	0	0	0	0

Table 6. Distribution of Protocol and Flags for Last Packets (Week 2).

Source IP	Protocol	TCP	UDP	ACK	PSH	RST	SYN	FIN	DON'T FRAG	MORE FRAG
66.102.7.99	TCP	1	0	1	0	0	0	0	0	0
216.228.2.120	DNS	0	1	0	0	0	0	0	0	0
216.228.2.120	DNS	0	1	0	0	0	0	0	0	0
63.245.209.31	TCP	1	0	1	0	0	0	0	0	0
66.35.214.30	TCP	1	0	1	0	0	0	0	0	0
207.46.216.56	TCP	1	0	1	0	0	0	0	0	0
207.46.216.62	TCP	1	0	1	0	0	0	0	0	0
128.241.21.146	TCP	1	0	1	0	1	0	0	0	0
216.73.86.52	TCP	1	0	1	0	0	0	0	0	0
216.73.86.91	TCP	1	0	1	0	0	0	0	0	0
216.228.2.120	DNS	0	1	0	0	0	0	0	0	0
207.46.130.100	NTP	0	1	0	0	0	0	0	0	0
207.46.209.126	TCP	1	0	1	0	1	0	0	0	0
65.55.192.29	TCP	1	0	1	0	1	0	0	0	0
207.46.212.62	TCP	1	0	1	0	1	0	0	0	0
131.107.115.28	TCP	1	0	1	0	1	0	0	0	0

Source IP	Protocol	TCP	UDP	ACK	PSH	RST	SYN	FIN	DON'T FRAG	MORE FRAG
207.46.13.30	TCP	1	0	1	0	1	0	0	0	0
207.46.221.222	TCP	1	0	1	0	1	0	0	0	0
207.46.13.28	TCP	1	0	1	0	1	0	0	0	0
66.77.84.82	TCP	1	0	1	0	1	0	0	0	0
216.228.2.120	DNS	0	1	0	0	0	0	0	0	0
66.77.84.82	TCP	1	0	1	0	1	0	0	0	0
207.46.212.62	TCP	1	0	1	0	1	0	0	0	0
216.228.2.120	DNS	0	1	0	0	0	0	0	0	0
Total		18	6	18	0	11	0	0	0	0

Table 7. Distribution of Protocol and Flags for Last Packets (Week 3).

Source IP	Protocol	TCP	UDP	ACK	PSH	RST	SYN	FIN	DON'T FRAG	MORE FRAG
66.102.7.99	TCP	1	0	1	0	0	0	0	0	0
216.228.2.120	DNS	0	1	0	0	0	0	0	0	0
216.228.2.120	DNS	0	1	0	0	0	0	0	0	0
210.51.23.237	Messenger	0	1	0	0	0	0	0	0	0
204.16.208.80	Messenger	0	1	0	0	0	0	0	0	0
195.27.116.145	Messenger	0	1	0	0	0	0	0	0	0
194.174.170.115	Messenger	0	1	0	0	0	0	0	0	0
204.16.208.52	Messenger	0	1	0	0	0	0	0	0	0
210.51.21.136	Messenger	0	1	0	0	0	0	0	0	0
207.46.130.100	NTP	0	1	0	0	0	0	0	0	0
202.97.238.196	Messenger	0	1	0	0	0	0	0	0	0
202.97.238.195	Messenger	0	1	0	0	0	0	0	0	0
220.164.140.249	Messenger	0	1	0	0	0	0	0	0	0
218.10.137.140	Messenger	0	1	0	0	0	0	0	0	0
207.46.209.126	TCP	1	0	1	0	1	0	0	0	0
194.145.63.131	Messenger	0	1	0	0	0	0	0	0	0
195.0.19.0	Messenger	0	1	0	0	0	0	0	0	0
194.221.241.74	Messenger	0	1	0	0	0	0	0	0	0
60.11.125.44	Messenger	0	1	0	0	0	0	0	0	0
221.209.110.48	Messenger	0	1	0	0	0	0	0	0	0
194.112.90.240	Messenger	0	1	0	0	0	0	0	0	0
204.16.208.75	Messenger	0	1	0	0	0	0	0	0	0
60.11.125.42	Messenger	0	1	0	0	0	0	0	0	0
194.239.12.110	Messenger	0	1	0	0	0	0	0	0	0
194.40.92.223	Messenger	0	1	0	0	0	0	0	0	0
202.97.238.201	Messenger	0	1	0	0	0	0	0	0	0
204.16.208.66	Messenger	0	1	0	0	0	0	0	0	0
221.208.208.92	Messenger	0	1	0	0	0	0	0	0	0
204.16.208.49	Messenger	0	1	0	0	0	0	0	0	0
221.208.208.98	Messenger	0	1	0	0	0	0	0	0	0
194.253.130.89	Messenger	0	1	0	0	0	0	0	0	0
194.216.135.144	Messenger	0	1	0	0	0	0	0	0	0
204.16.209.20	Messenger	0	1	0	0	0	0	0	0	0

Source IP	Protocol	TCP	UDP	ACK	PSH	RST	SYN	FIN	DON'T FRAG	MORE FRAG
207.46.212.62	TCP	1	0	1	0	1	0	0	0	0
207.46.253.157	TCP	1	0	1	0	1	0	0	0	0
204.16.208.23	Messenger	0	1	0	0	0	0	0	0	0
202.97.238.202	Messenger	0	1	0	0	0	0	0	0	0
221.209.110.47	Messenger	0	1	0	0	0	0	0	0	0
221.208.208.103	Messenger	0	1	0	0	0	0	0	0	0
221.10.224.253	Messenger	0	1	0	0	0	0	0	0	0
221.208.208.212	Messenger	0	1	0	0	0	0	0	0	0
221.208.208.90	Messenger	0	1	0	0	0	0	0	0	0
202.97.238.199	Messenger	0	1	0	0	0	0	0	0	0
221.208.208.99	Messenger	0	1	0	0	0	0	0	0	0
221.209.110.49	Messenger	0	1	0	0	0	0	0	0	0
60.11.125.54	Messenger	0	1	0	0	0	0	0	0	0
216.228.2.120	DNS	0	1	0	0	0	0	0	0	0
221.208.208.83	Messenger	0	1	0	0	0	0	0	0	0
Total		4	44	4	0	3	0	0	0	0

Table 8. Distribution of Protocol and Flags for Last Packets (Week 4).

We were, however, unable to observe any other trends of interest, except for the unusually high User Datagram Protocol (UDP) packets on Week 4, despite the reduction of network traffic, as mentioned in Section B of this Chapter. We correlated these UDP packets with their sizes and realized that they were responsible for two other modal regions where packet sizes ranged within 500-600 and 900-950 bytes. We checked associated source IP addresses of all last packets received by our honeypots and realized that our previous observation of packet sizes between 50-100 bytes belonged to legitimate traffic generated during the set-up of our honeypots. They were from organizations like Google, Microsoft Corporation, and Redshift (that hosted our DNS server). Further packet inspections revealed no malicious intent. This, had, however, masked the potential signatures of the other two modal regions. Table 9 shows the organization names and countries of location of the source IP addresses responsible for the last packets in Week 4. Deep inspection of packets with Messenger (UDP) protocol revealed malicious intent, since they attepted to persuade users to download Windows registry updates when they were not authoritative sources. The intent might be to decept

unwary user to assist the cyber-attackers to download root-kits from the designated websites. Figures 19 and 20 show content of malicious intent in packets responsible for the 500-600 and 900-950 bytes modal regions, respectively.

Source IP	Protocol	Size	Organization Name	Location
66.102.7.99	TCP	60	GOOGLE	USA
216.228.2.120	DNS	239	REDSHIFT	USA
216.228.2.120	DNS	334	REDSHIFT	USA
210.51.23.237	Messenger	501	CNCNET-CN	CHINA
204.16.208.80	Messenger	557	FAST-COLOCATION	USA
195.27.116.145	Messenger	922	CW	SPAIN
194.174.170.115	Messenger	922	AS702	GERMANY
204.16.208.52	Messenger	557	FAST-COLOCATION	USA
210.51.21.136	Messenger	501	CNCNET-CN	CHINA
207.46.130.100	NTP	90	MICROSOFT CORP	USA
202.97.238.196	Messenger	499	CHINA169-BACKBONE	CHINA
202.97.238.195	Messenger	500	CHINA169-BACKBONE	CHINA
220.164.140.249	Messenger	942	CHINA169-BACKBONE	CHINA
218.10.137.140	Messenger	500	CHINA169-BACKBONE	CHINA
207.46.209.126	TCP	60	MICROSOFT CORP	USA
194.145.63.131	Messenger	922	DIRBG-AS	BULGARIA
195.0.19.0	Messenger	922	SCARLET	BELGIUM
194.221.241.74	Messenger	922	CW	GERMANY
60.11.125.44	Messenger	940	CHINA169-BACKBONE	CHINA
221.209.110.48	Messenger	499	CHINA169-BACKBONE	CHINA
194.112.90.240	Messenger	922	CW	GERMANY
204.16.208.75	Messenger	458	FAST-COLOCATION	USA
60.11.125.42	Messenger	942	CHINA169-BACKBONE	CHINA
194.239.12.110	Messenger	922	TDC	DENMARK
194.40.92.223	Messenger	922	UNSPECIFIED	SWITZERLAND
202.97.238.201	Messenger	500	CHINA169-BACKBONE	CHINA
204.16.208.66	Messenger	459	FAST-COLOCATION	USA
221.208.208.92	Messenger	499	CHINA169-BACKBONE	CHINA
204.16.208.49	Messenger	459	FAST-COLOCATION	USA
221.208.208.98	Messenger	499	CHINA169-BACKBONE	CHINA
194.253.130.89	Messenger	922	IANA-RSVD-0	DENMARK
194.216.135.144	Messenger	922	AS702	UK
204.16.209.20	Messenger	458	FAST-COLOCATION	USA
207.46.212.62	TCP	60	MICROSOFT CORP	USA
207.46.253.157	TCP	60	MICROSOFT CORP	USA
204.16.208.23	Messenger	459	FAST-COLOCATION	USA
202.97.238.202	Messenger	500	CHINA169-BACKBONE	CHINA
221.209.110.47	Messenger	500	CHINA169-BACKBONE	CHINA
221.208.208.103	Messenger	499	CHINA169-BACKBONE	CHINA
221.10.224.253	Messenger	955	CHINA169-BACKBONE	CHINA
221.208.208.212	Messenger	499	CHINA169-BACKBONE	CHINA

Source IP	Protocol	Size	Organization Name	Location
221.208.208.90	Messenger	501	CHINA169-BACKBONE	CHINA
202.97.238.199	Messenger	500	CHINA169-BACKBONE	CHINA
221.208.208.99	Messenger	499	CHINA169-BACKBONE	CHINA
221.209.110.49	Messenger	500	CHINA169-BACKBONE	CHINA
60.11.125.54	Messenger	500	CHINA169-BACKBONE	CHINA
216.228.2.120	DNS	501	REDSHIFT	USA
221.208.208.83	Messenger	500	CHINA169-BACKBONE	CHINA

Table 9. Organization Name and Location of Source IP Addresses for Week 4.

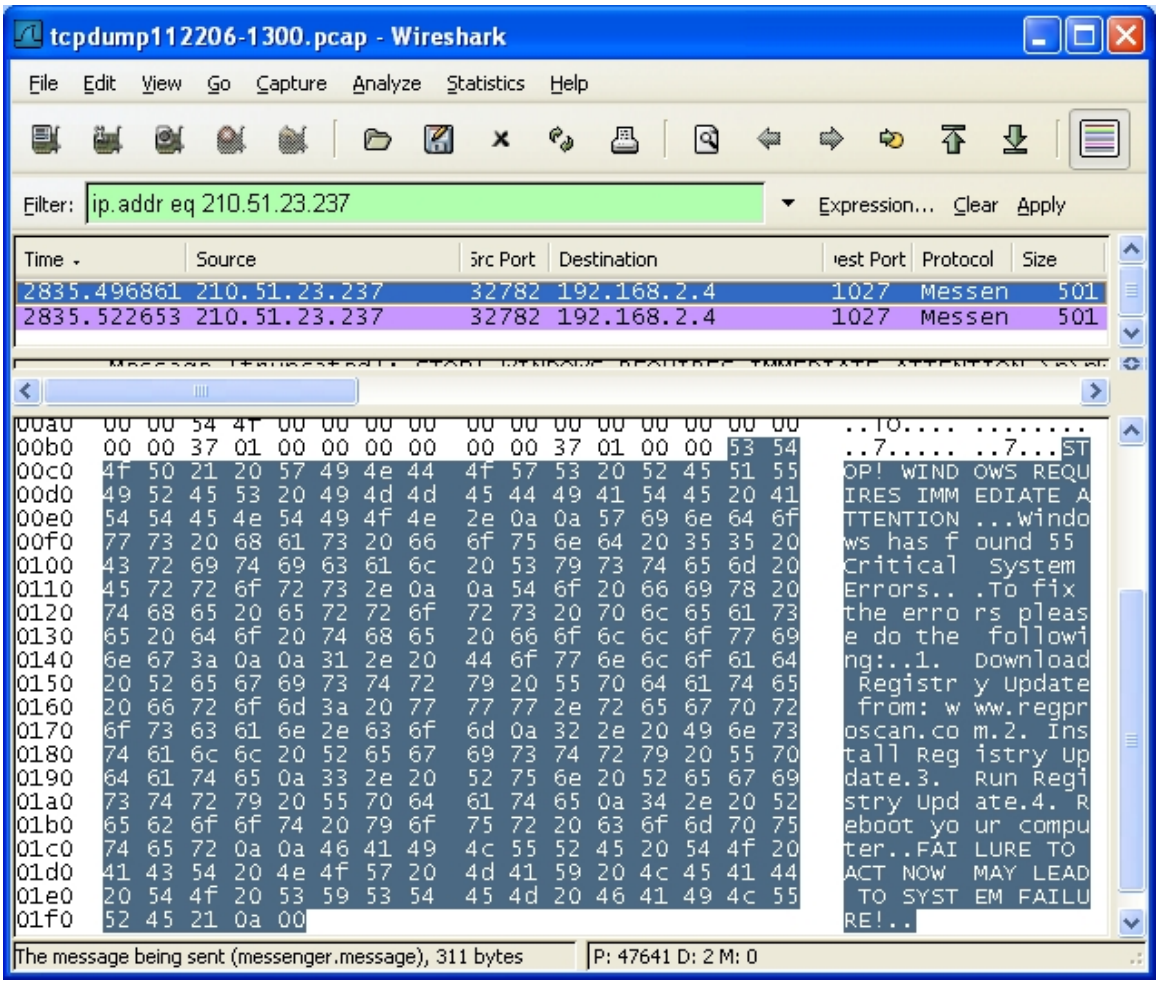

Figure 19. Screen Capture of Packet Inspection for Packet from IP Address 210.51.23.237 (Packet Size = 501).

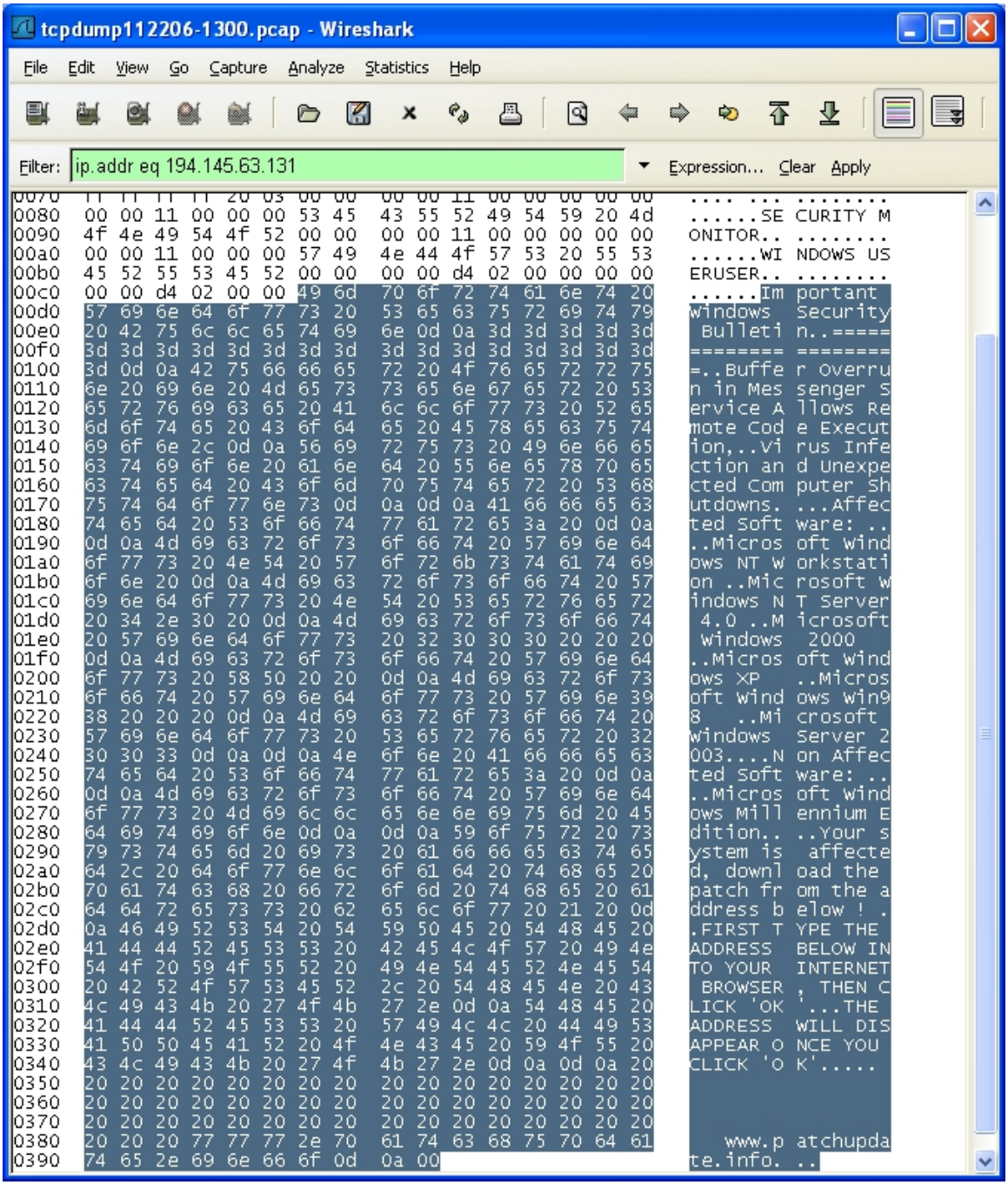

Figure 20. Screen Capture of Packet Inspection for Packet from IP Address 194.145.63.131 (Packet Size = 922).

Figure 21 shows the distribution of the sizes of malicious Messenger (UDP) protocol packets. We observed that frequencies of packet size of (500 ± 1) and 922 bytes were significantly higher. This might indicate that these two packets were more popular with the cyber-attackers and would serve as good signatures. These attacks,

however, were passive and would require cooperation of users. In our context, our honeypot may eventually frustrate the cyber-attackers as they gradually learn that we do not intend to cooperate with them (in the downloading of root-kits).

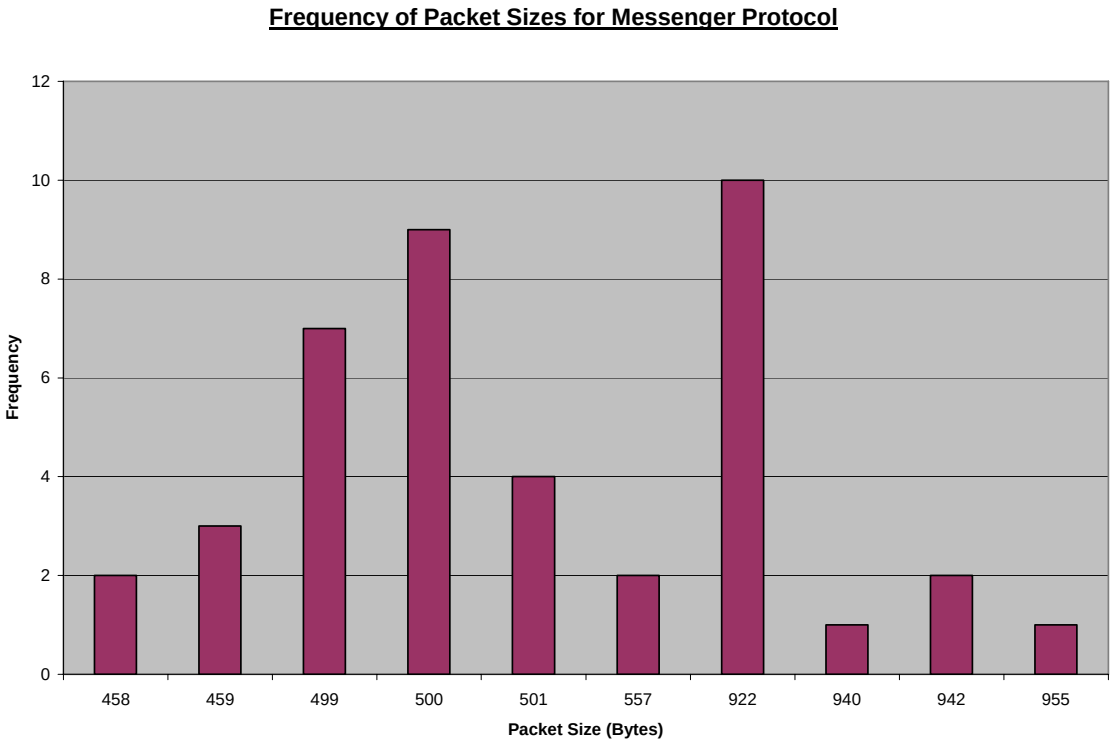

Figure 21. Frequency Plot of Malicious Packet Sizes.

VI. CONCLUSIONS

> The elementary principle of all deception is to attract the enemy's attention to what you wish him to see and to distract his attention from what you do not wish him to see. It is by these methods that the skilful conjuror obtains his results. –
>
> Memo to Chiefs of Staff, 1940 from General Wavell, Middle East Commander (1939-41)

A. CONCLUSIONS

We began the investigation to try to confirm the existence of avoidance behavior induced by a honeynet. We intend to exploit this factor on real production systems to deceive cyber-attackers to avoid further reconnaissance and compromise. We based our investigation on the assumption of some awareness and availability of anti-honeypot technology among the cyber-attacker community. As we did not see evidence of exploitation after active reconnaissance, our fake honeynet setup did at least withstand exposure to the cyber-attackers for 28 days.

While camouflage does not provide physical protection to the soldiers against hostile rounds, it offers concealment through obscurity. The deployment of a fake honeynet aims to achieve similar effects by concealing our real production systems from cyber-attackers. In addition, the fake honeynet serves as protective mimicry when it encounters a cyber-attacker, much as how some butterflies create confusing patterns on their wings that look like eyes to fool predators. We believe that widespread deployment of fake honeynets amidst real honeynets may create further confusion for cyber-attackers and delay or impede their attacks. It should be noted that security administrators should not solely rely on fake honeypots for protection, but skillfully use it as one component in the defense-in-depth information security strategy.

The protection of information system is a broad and complex problem as opposed to the relative simplicity of a successful exploitation. As such, security professionals have been lagging in the race against the hackers. With the introduction of honeynets and fake honeynets we can reverse the security power balance. Professional hackers will

have to keep pace with the honeynet development and devote resources to develop anti-honeynet technology or risk their prized techniques of exploitation and a higher chance of being caught and prosecuted.

B. APPLICATIONS

Fake honeynets can be extended to Rapid Response Command and Control (R2C2) systems, scalable communications for regional combatant commanders developed by the Deployable Joint Command and Control (DJC2) (SEA-9 R2C2 Team, 2006). The self-contained nature of a fake honeynet makes it a suitable lightweight candidate for R2C2 systems while providing a two-man team adequate obscurity in potentially hostile deployed environment.

C. FUTURE WORKS

A goal of this thesis assesses the effects of honeynet on cyber-attackers. Despite our IP address advertisements, our fake honeynet received significantly lower traffic volume. It may be attributed to duration of exposure. While we have deployed a high-interaction honeynet, our collection mechanism still relies on the initiatives of cyber-attackers to explore and exploit. To enhance the rate of solicitation and clarity in the cause-and-effect, we will need to be more active in our interaction. We suggest the following future investigations

1. Tailored Responses to Cyber-attackers. We need to log reactive behaviors of cyber-attackers to our various specific responses to their reconnaissance. This will provide better clarity on effects of deterring responses. This can be explored with Snort_inline.

2. In-depth Exploration and Exploitation. In the next phase, cyber-attackers should be allowed easy access into the depth of the honeynets. This will allow us to understand the exploration and exploitation mechanism as well as the reactions (to honeynet) beyond the reconnaissance phase.

3. Lightweight, High-interaction Honeynets. We need to continue to design lightweight, high-interaction honeynets. This will facilitate easy deployment of honeynet. In addition, it can be installed on real production systems and function as a honeynet inoculation.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. RESULT PLOTS

A. RESULTS FROM SESSION ANALYSIS

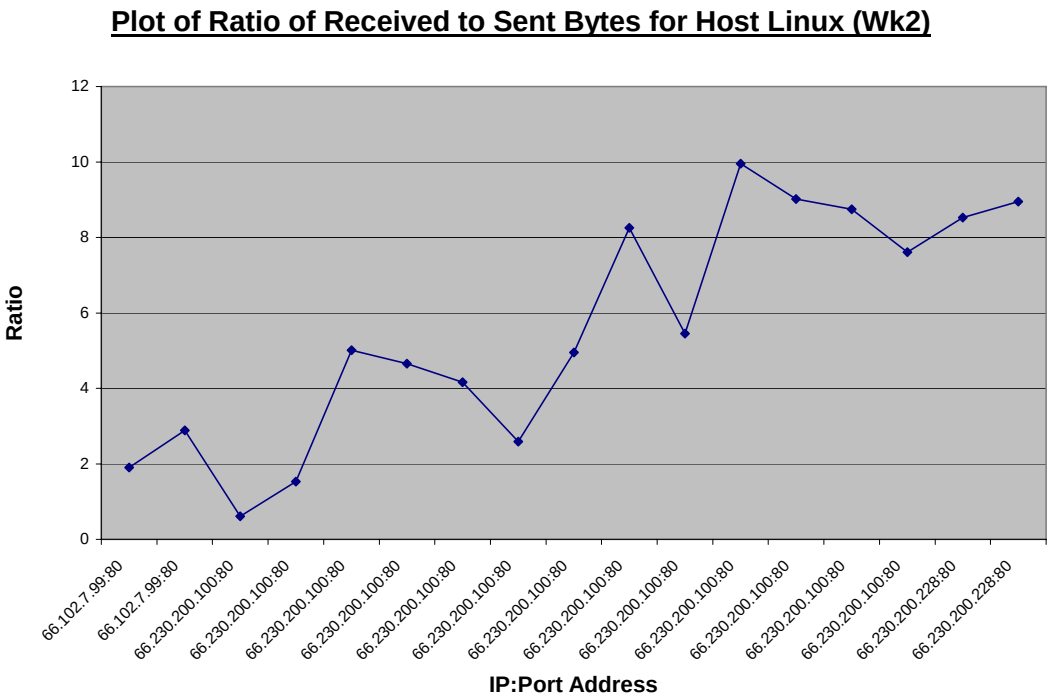

Figure 22. Plot of Ratio of Received to Sent Bytes for Host Linux (Week 2).

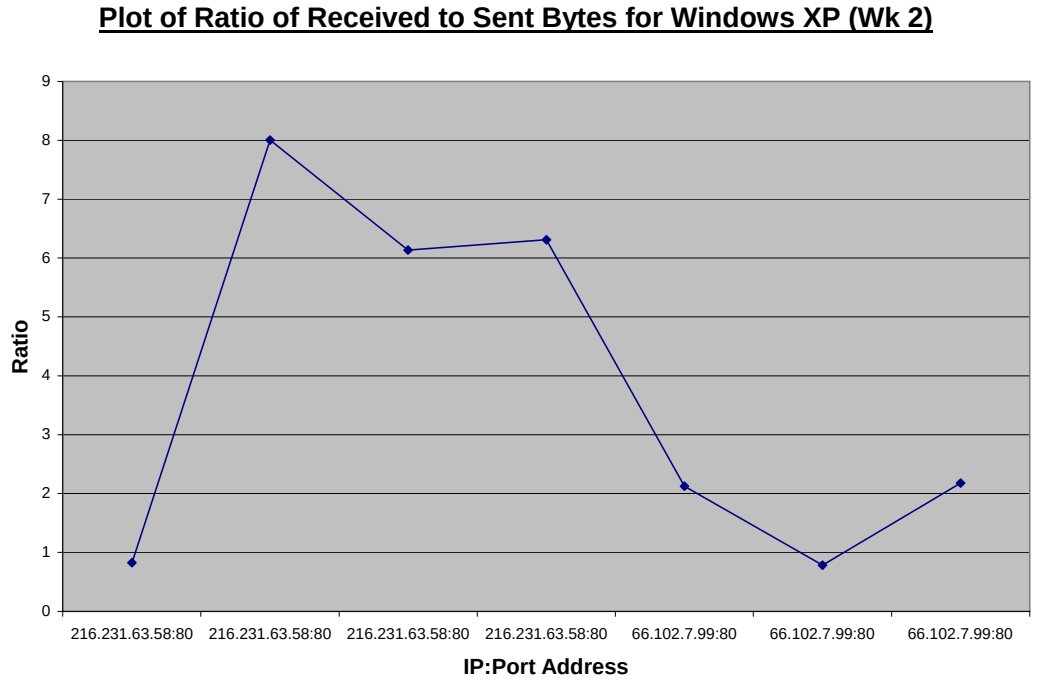

Figure 23. Plot of Ratio of Received to Sent Bytes for Windows XP (Week 2).

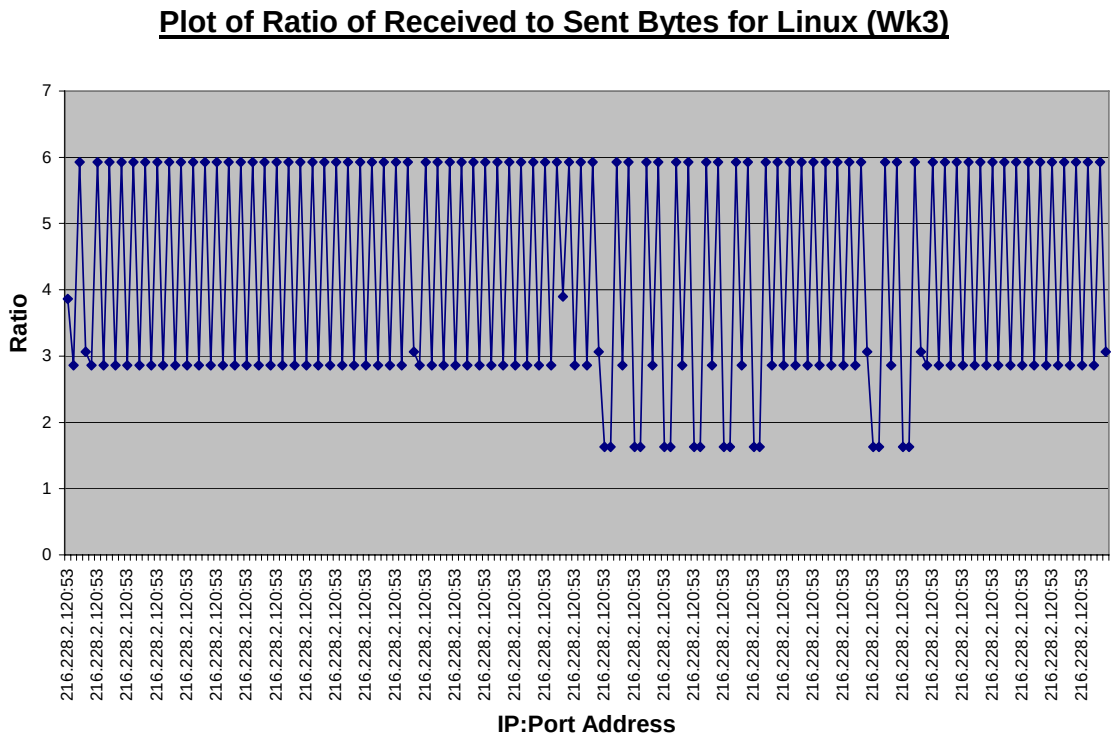

Figure 24. Plot of Ratio of Received to Sent Bytes for Linux (Week 3).

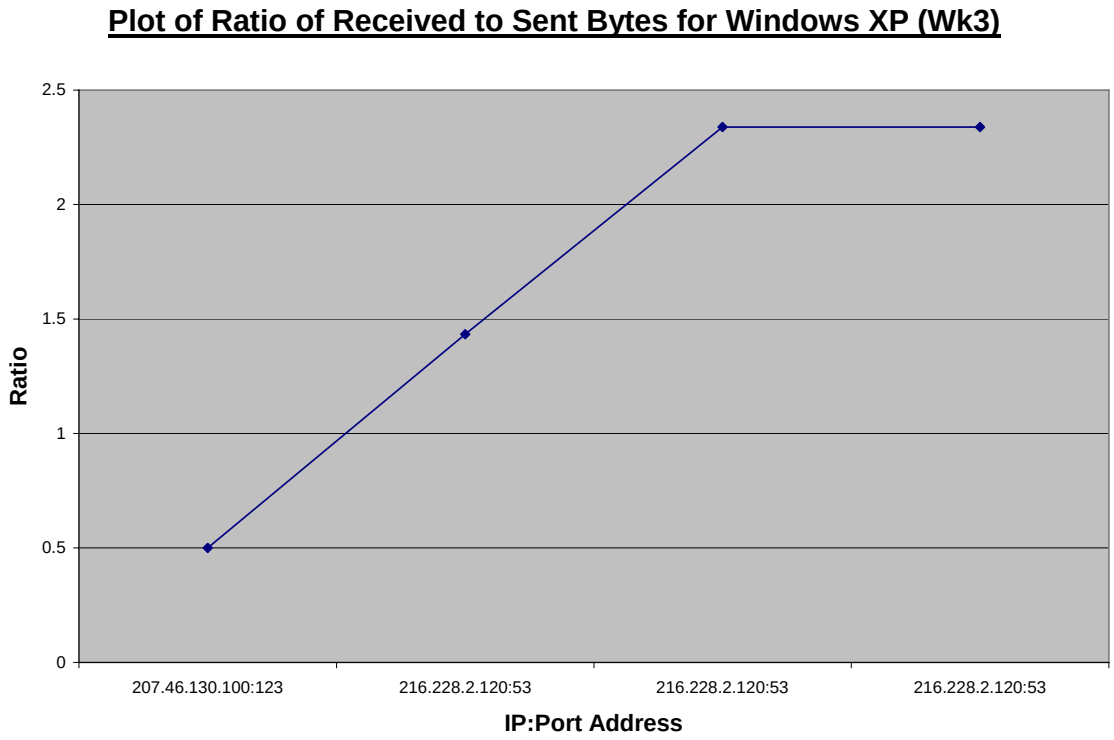

Figure 25. Plot of Ratio of Received to Sent Bytes for Windows XP (Week 3).

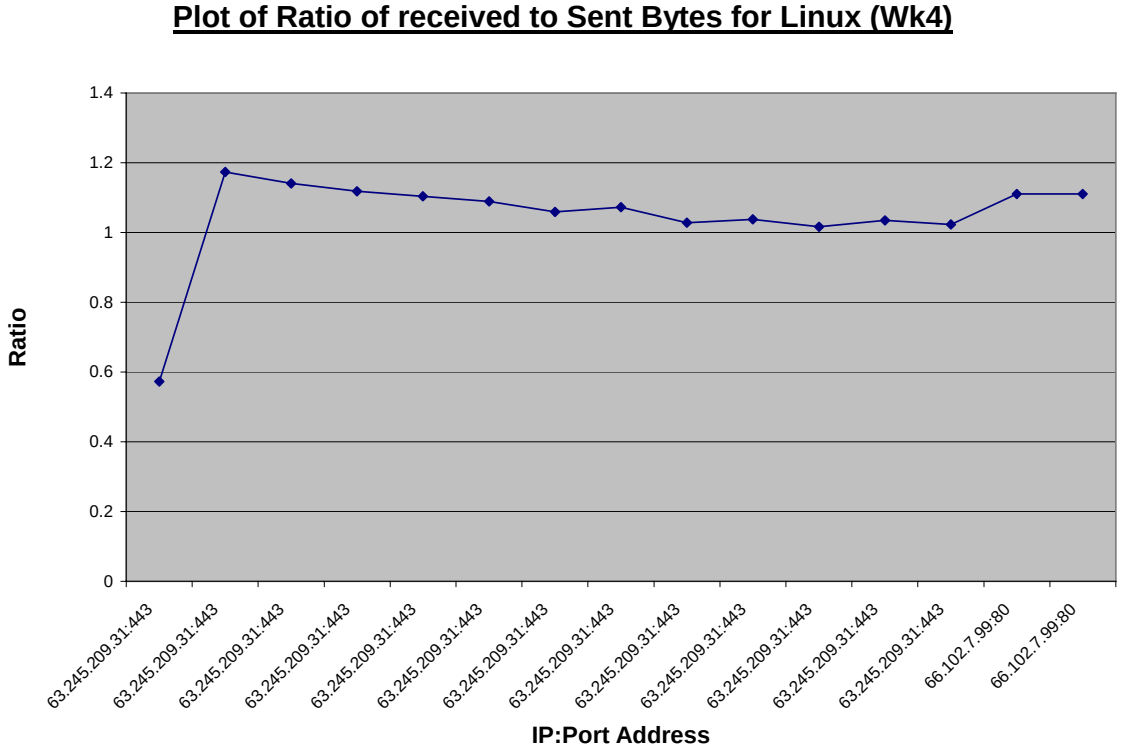

Figure 26. Plot of Ratio of Received to Sent Bytes for Linux (Week 4).

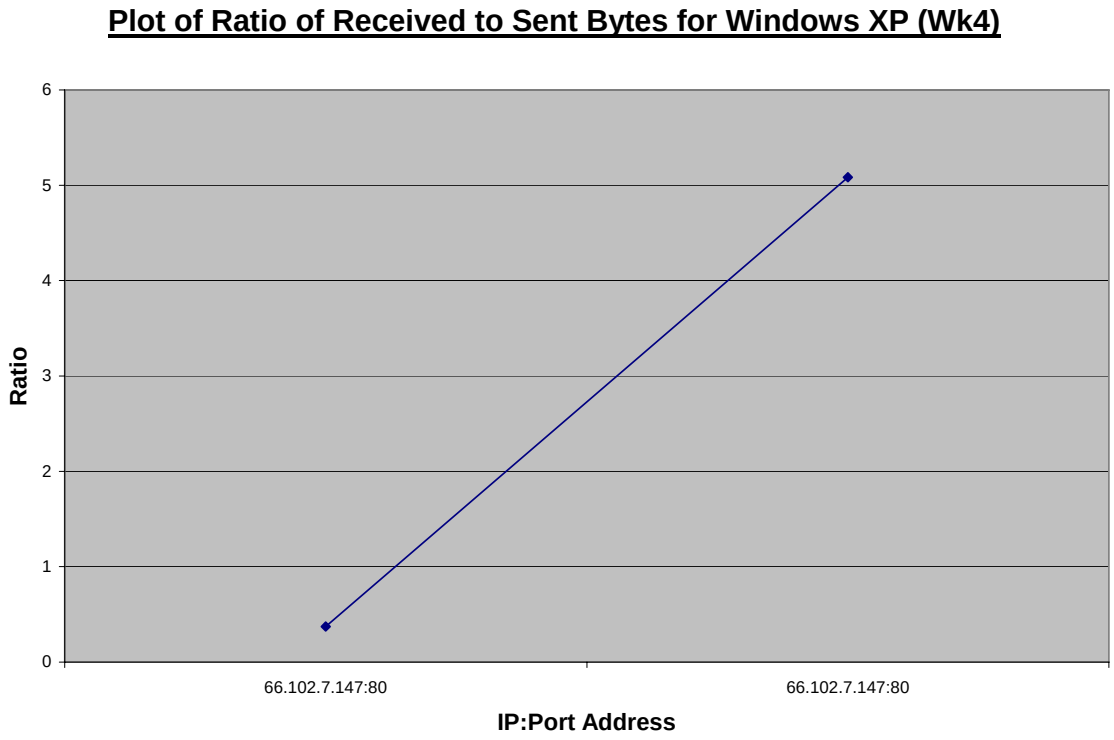

Figure 27. Plot of Ratio of Received to Sent Bytes for Windows XP (Week 4).

B. RESULTS FROM TIME DOMAIN ANALYSIS

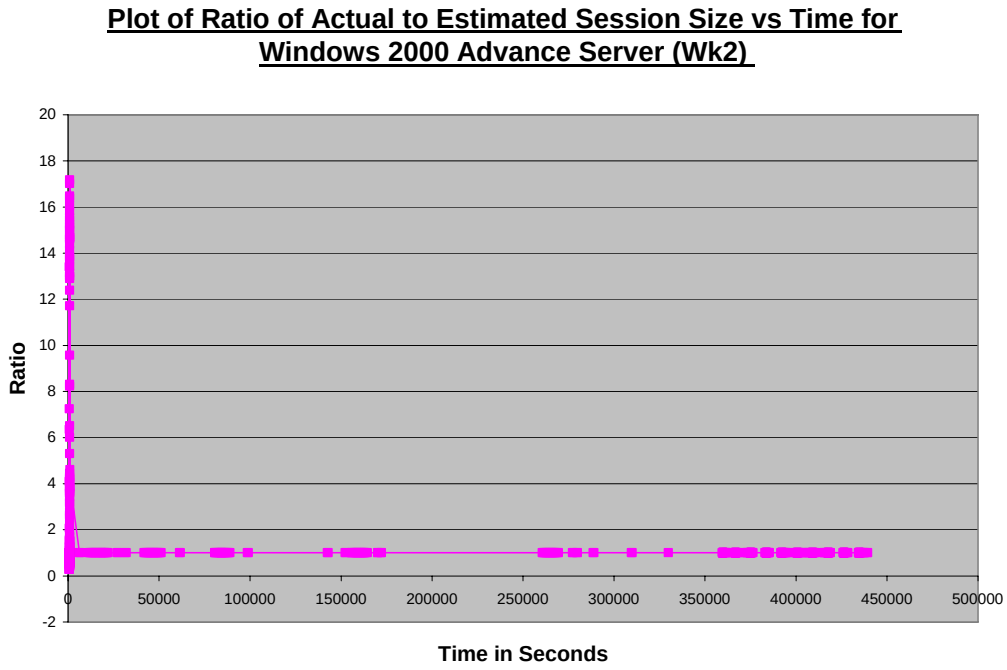

Figure 28. Plot of Ratio of Actual to Estimated Size against Time for Windows 2000 Advanced Server Operating System (Week 2).

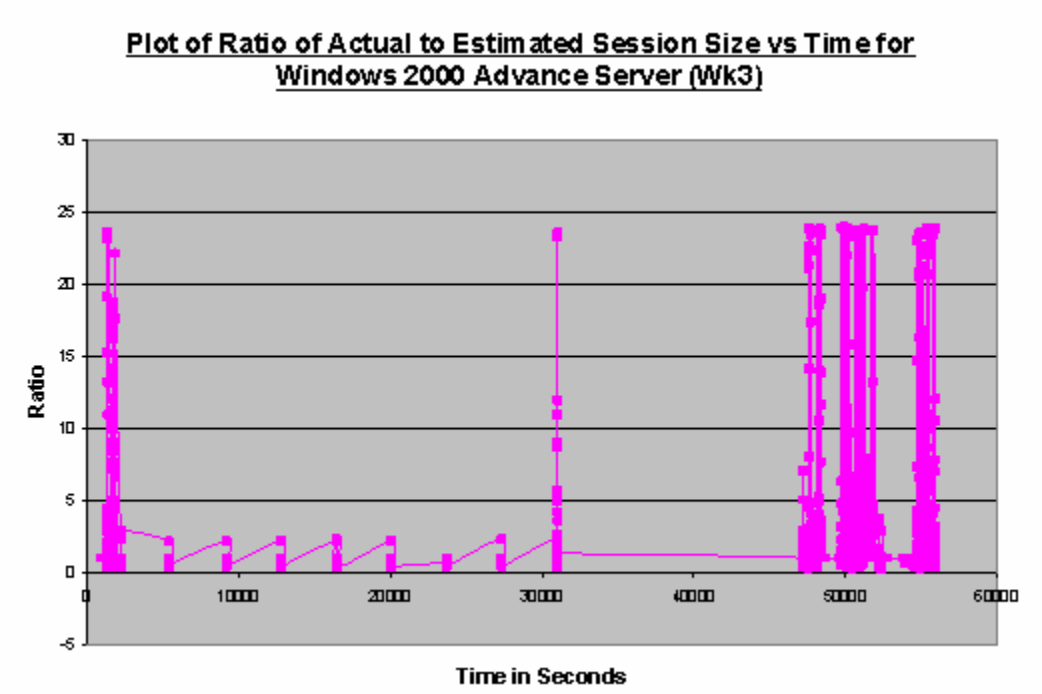

Figure 29. Plot of Ratio of Actual to Estimated Size against Time for Linux Host Operating System (Week 3).

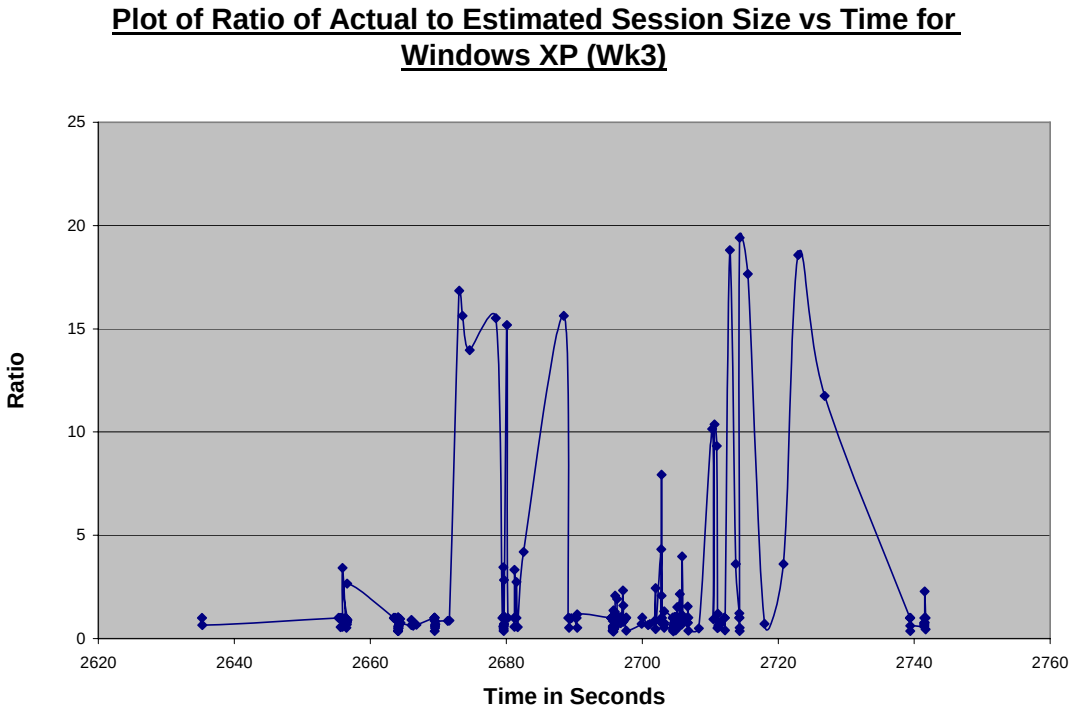

Figure 30. Plot of Ratio of Actual to Estimated Size against Time for Windows XP Operating System (Week 3).

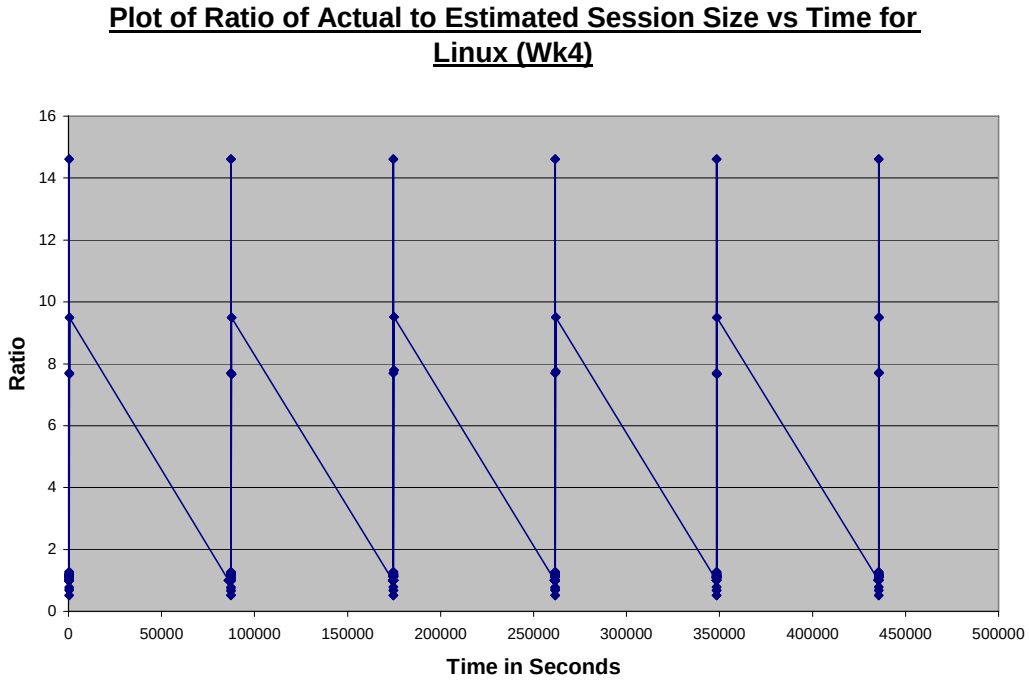

Figure 31. Plot of Ratio of Actual to Estimated Size against Time for Linux Host Operating System (Week 4).

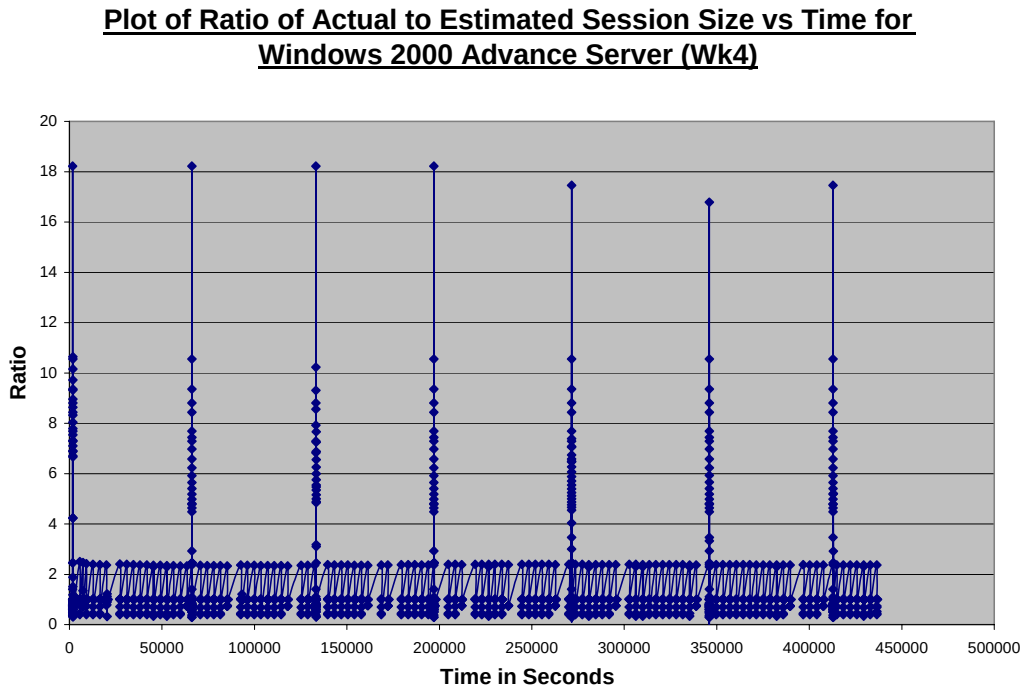

Figure 32. Plot of Ratio of Actual to Estimated Size against Time for Windows 2000 Advanced Server Operating System (Week 4).

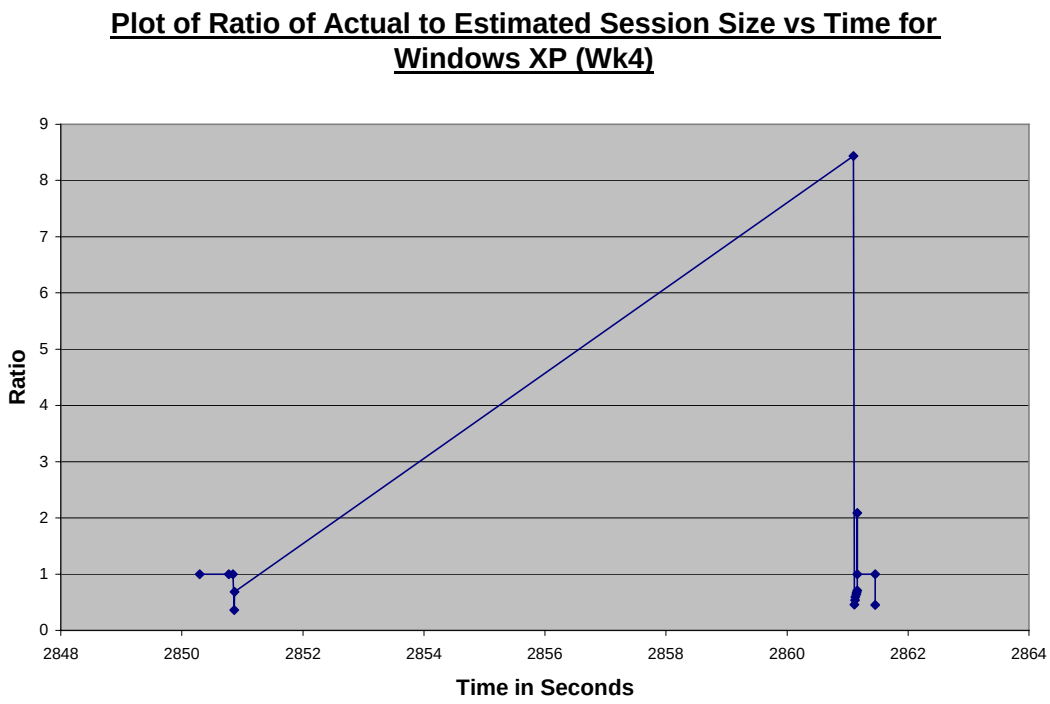

Figure 33. Plot of Ratio of Actual to Estimated Size against Time for Windows XP Operating System (Week 4).

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. SOURCE CODES

```
/* Obtain the information for Tcpdump and generate the sum(size) and
//count(packet) for all socket pairs across time.
// Name tcpdumpAnalysisSocketPairSizeCountTime.java
// Author@ Harry Lim
// Created on November 28, 2006, 12:21 PM


import java.io.*;
import java.util.*;

class TcpdumpAnalysisSocketPairSizeCountTime {
  public static void main (String args[]) throws IOException {

      String Input, filename, ipAddress;
        int index= 0;
        int max = 500000;
        boolean update;
        int [] size;
        double [] time;
        String ia, garbage;
        String [] srcIP, srcPort, destIP, destPort, protocol;
        StringTokenizer str;

        //Connection database
        int i,j,jMax,k;
        int [] sizeCurrent, sumSizeCurrent, connectionCount;
        double [] timeCurrent;
        String [] srcIPCurrent, srcPortCurrent, destIPCurrent,
destPortCurrent, protocolCurrent;

        //Initialise input data array
        size = new int [max];
        time = new double [max];
        srcIP = new String [max];
        srcPort = new String [max];
        destIP = new String [max];
        destPort = new String [max];
        protocol =  new String [max];

        //Initialise storage data array
        sumSizeCurrent = new int [max];
        connectionCount = new int [max];
        sizeCurrent = new int [max];
        timeCurrent = new double [max];
        srcIPCurrent = new String [max];
        srcPortCurrent = new String [max];
        destIPCurrent = new String [max];
        destPortCurrent = new String [max];
        protocolCurrent =  new String [max];


        //Reading from Tcpdump txt file
```

```
        filename = args[0];

          System.out.println("Enter Honeypot IP Address: ");
          BufferedReader in = new BufferedReader(new
          InputStreamReader(System.in));
          ia = in.readLine();

        FileReader fr = new FileReader(filename);
        BufferedReader br = new BufferedReader(fr);
        //Start-To be removed; for debugging only
//      PrintWriter fileout = new PrintWriter(new FileWriter("debug-
verify-initialization.txt"));
        //End-To be removed.
        while ((Input = br.readLine()) != null) {
            str = new StringTokenizer (Input);
              if (str.countTokens() == 9)
              {
//                  System.out.println("Error: Wrong number of
tokens");
//                  return;
//              }
              garbage  = str.nextToken(); //digest serial number token
              time [index] = Double.parseDouble(str.nextToken());
              srcIP [index] = str.nextToken();
              srcPort [index] = str.nextToken();
              destIP [index] = str.nextToken();
              destPort [index] = str.nextToken();
              protocol [index] = str.nextToken();
              size [index] = Integer.parseInt(str.nextToken());
//            System.out.println("SourceIP: "+srcIP[index]+" \t
SourcePort: "+srcPort[index]+" \t Dest IP: "+destIP[index]+" \t
destPort: "+destPort[index]+" \t Protocol: "+protocol[index]+" \t size:
"+size[index]"\n");
//              fileout.println("Time: " +time[index]+"SourceIP:
"+srcIP[index]+" \t SourcePort: "+srcPort[index]+" \t Dest IP:
"+destIP[index]+" \t destPort: "+destPort[index]+" \t Protocol:
"+protocol[index]+" \t size: "+size[index]);
              System.out.println("Index: "+index);
              index ++;
                    }
          }

////////////////////////////////////////////////////////////////////
////////////////////
// Socket Pair Filtering:
// - determine connection between two socket pairs with designated IP
address,
// - record their time
// - record their size
// - record their sum of size (to current time)

// i is the counter for the raw database.
// jMax is the maximum number of connections.
          jMax = 0;
// connectionCount[jMax] is the counter of the time-IPsocketPair-bytes
Database
          for (i=0; i < index; i++){
```

```
                update = false;
//                System.out.println("IP Address:
"+ia+"SrcIP"+srcIP[i]+"DestIP"+destIP[i]);
            if ((srcIP[i].equals(ia)) || (destIP[i].equals(ia))){
//Initialized interested database for specific IP address (of honeynet)
                        timeCurrent [jMax] = time [i];
//
System.out.println("True!"+timeCurrent[jMax]+"\t time = "+ time[i]);
                        srcIPCurrent [jMax] = srcIP[i];
                        srcPortCurrent [jMax] = srcPort[i];
                        destIPCurrent [jMax] = destIP[i];
                        destPortCurrent [jMax] = destPort[i];
                        protocolCurrent [jMax] = protocol[i];
                        sizeCurrent [jMax] = size[i];
// j is the counter for the connection database
                        for (j = 0; j < jMax; j++){
                            if (srcIP[i].equals(srcIPCurrent[j]) &&
destIP[i].equals(destIPCurrent[j]) &&
srcPort[i].equals(srcPortCurrent[j]) &&
destPort[i].equals(destPortCurrent[j])){
// This implies an older ip socket pair exists.  Update the sum of size
and count for that connection.
                                sumSizeCurrent[jMax]                    =
sumSizeCurrent[j]+sizeCurrent[jMax];
                                connectionCount[jMax]                   =
connectionCount[j]+1;
                                update = true;
                            }

                        }
// This implies a new ip pair since the search is always in strictly
incremental time.

                        if (update != true){
                            sumSizeCurrent [jMax] = size[i];
                            connectionCount [jMax]= 1;
                                        }
                        jMax++;
                }
          }



//////////////////////////////////////////////////////////////////////
//////////////////////

//////////////////////////////////////////////////////////////////////
//////////////////////
//Print the Last packets to file:
        PrintWriter    socketPairSizeCount    =    new    PrintWriter(new
FileWriter("socketPairSizeCount"+filename+ia+".txt"));
        for (k = 0; k < jMax; k++){
 //         socketPairSizeCount.println(k+"\t SN: "+ snLast[k]+"\t time:
"   +   timeLast[k]+"   \t   source:   "+sourceLast[k]+"   \t   srcport:
"+srcportLast[k]+"    \t    dest:    "+destLast[k]+"    \t    destport:
"+destportLast[k]+"   \t   protocol:   "+protocolLast[k]+"   \t   size:
"+sizeLast[k]+"\t CumSize: "+cumSize[index]+"\n");
```

```
        socketPairSizeCount.println(timeCurrent[k]+"
\t"+srcIPCurrent[k]+"   \t"+srcPortCurrent[k]+"   \t"+destIPCurrent[k]+"
\t"+destPortCurrent[k]+"  \t"+protocolCurrent[k]+"  \t"+sizeCurrent[k]+"
\t"+sumSizeCurrent[k]+"\t"+connectionCount[k]);
  }
//        socketPairSizeCount.println ("\nTotal Last Packets = "+jMax);
        socketPairSizeCount.close();
////////////////////////////////////////////////////////////////////////
////////////////////
        fr.close();
        //Start-To be removed; for debugging only
//      fileout.close();
        //End-To be removed.

  }
}
```

```
//Obtain meta-data of Last packet of the connection between two
//machinces.
//Name tcpdumpAnalysisConnectionLastPacket.java
// Author@ Harry Lim
// Created on November 21, 2006, 12:21 PM


import java.io.*;
import java.util.*;

class TcpdumpAnalysisConnectionLastPacket {
  public static void main (String args[]) throws IOException {

      String Input, filename;
        int index= 0;
        int max = 500000;
        int [] sn, size, cumSize;
        double [] time;
        String [] source, srcport, dest, destport, protocol;
        StringTokenizer str;

        //Last packet database
        int i,j,jMax,k, update;
        int indexLast = 0;
        int maxLast = 500000;
        int [] snLast, sizeLast, cumSizeLast;
        double [] timeLast;
            String garbage;
        String  []  sourceLast,  srcportLast,  destLast,  destportLast,
protocolLast;



        //Initialise data array
        size = new int [max];
        time = new double [max];
        source = new String [max];
        srcport = new String [max];
        dest = new String [max];
        destport = new String [max];
        protocol =  new String [max];

        //Initialise Last packet data array

        sizeLast = new int [max];
        cumSizeLast= new int [max];
        timeLast = new double [max];
        sourceLast = new String [max];
        srcportLast = new String [max];
        destLast = new String [max];
        destportLast = new String [max];
        protocolLast =  new String [max];


        //Reading from Tcpdump txt file

        filename = args[0];
```

```
            FileReader fr = new FileReader(filename);
        BufferedReader br = new BufferedReader(fr);
        while ((Input = br.readLine()) != null) {
            str = new StringTokenizer (Input);
              if (str.countTokens() == 9)
              {
//                            System.out.println("Error: Wrong number of
tokens");
//                  return;
//              }
              garbage  = str.nextToken();
              time [index] = Double.parseDouble(str.nextToken());
              source [index] = str.nextToken();
              srcport [index] = str.nextToken();
              dest [index] = str.nextToken();
              destport [index] = str.nextToken();
              protocol [index] = str.nextToken();
              size [index] = Integer.parseInt(str.nextToken());
//            cumSize[index] = Integer.parseInt(str.nextToken());
//                 System.out.println("SN: "+ sn[index]+"\t time: " +
time[index]+"    \t    source:    "+source[index]+"    \t    srcport:
"+srcport[index]+"    \t    dest:    "+dest[index]+"    \t    destport:
"+destport[index]+"   \t   protocol:   "+protocol[index]+"   \t   size:
"+size[index]+"\t CumSize: "+cumSize[index]+"\n");
//                   fileout.println("SN: "+ sn[index]+"\t time: " +
time[index]+"    \t    source:    "+source[index]+"    \t    srcport:
"+srcport[index]+"    \t    dest:    "+dest[index]+"    \t    destport:
"+destport[index]+"   \t   protocol:   "+protocol[index]+"   \t   size:
"+size[index]+"\t CumSize: "+cumSize[index]+"\n");
              System.out.println("Time: "+ time[index]);
              index ++;
                    }
          }

////////////////////////////////////////////////////////////////////////
////////////////////
// Last packet analysis: determine last packet connection between two
ip address.

// i is the counter for the raw database.
          i = 0;
// j is the counter for the Last packet database.
          j = 0;
// jMax is the maximum number of last packets.
          jMax = 0;
          for (i=0; i < index; i++){
              update = 0;
              for (j=0; j < jMax; j++){
                  if    (((    source[i].equals(sourceLast[j])    &&
dest[i].equals(destLast[j]))  ||  (  source[i].equals(destLast[j])  &&
dest[i].equals(sourceLast[j]))) && (time [i] >= timeLast [j])){
// This implies an older ip pair exists.
                     timeLast [j] = time [i];
                     sourceLast [j] = source[i];
                     srcportLast [j] = srcport[i];
                     destLast [j] = dest[i];
                     destportLast [j] = destport[i];
```

```
                    protocolLast [j] = protocol[i];
                    sizeLast [j] = size[i];
                    cumSizeLast [j] = cumSizeLast[j] + size[i];
                    update = 1;
                  //populate the rest of the data less the source and
dest ip.
                  }
              }
              if ((update != 1)){
// This implies a new ip pair since the search is always in strictly
incremental time.
                    timeLast [jMax] = time [i];
                    sourceLast [jMax] = source[i];
                    srcportLast [jMax] = srcport[i];
                    destLast [jMax] = dest[i];
                    destportLast [jMax] = destport[i];
                    protocolLast [jMax] = protocol[i];
                    sizeLast [jMax] = size[i];
                    cumSizeLast [jMax] = size [i];
                    jMax++;
                }
                }

//////////////////////////////////////////////////////////////////////
////////////////////

//////////////////////////////////////////////////////////////////////
////////////////////
//Print the Last packets to file:
        PrintWriter     lastPackets     =     new     PrintWriter(new
FileWriter("lastPacket"+filename+".txt"));
        for (k = 0; k < jMax; k++){
 //          lastPackets.println(k+"\t SN: "+ snLast[k]+"\t time: " +
timeLast[k]+"    \t    source:    "+sourceLast[k]+"    \t    srcport:
"+srcportLast[k]+"    \t    dest:    "+destLast[k]+"    \t    destport:
"+destportLast[k]+"   \t   protocol:   "+protocolLast[k]+"   \t   size:
"+sizeLast[k]+"\t CumSize: "+cumSize[index]+"\n");
        lastPackets.println("\t"  +  timeLast[k]+"  \t"+sourceLast[k]+"
\t"+srcportLast[k]+"                                    \t"+destLast[k]+"
\t"+destportLast[k]+"\t"+protocolLast[k]+"\t"+sizeLast[k]+"\t"+cumSizeL
ast[k]);
  }
//        lastPackets.println ("\nTotal Last Packets = "+jMax);
        lastPackets.close();
//////////////////////////////////////////////////////////////////////
////////////////////
        fr.close();
//    fileout.close();
  }
}
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

1. Boyd, John (September 1976). *Destruction & Creation*. Retrieved December 2006 from http://www.belisarius.com/modern_business_strategy/boyd/destruction/destruction_and_creation.htm.

2. Duong, Binh T. (March 2006). M.S. thesis, Naval Postgraduate School: *Comparisons of Attacks on Honeypots with Those on Real Networks*. Retrieved December 2006 from www.cs.nps.navy.mil/people/faculty/rowe/oldstudents/duong_thesis.htm.

3. The Honeynet Project. (2004). *Know Your Enemy: Learning about Security Threats, Second Edition*, Boston, MA: Addison-Wesley.

4. Jones, Keith J., Bejtlich, Richard, Rose, Curtis W. (2006). *Real Digital Forensics: Computer Security and Incident Response*. Upper Saddle River, NJ: Addison-Wesley.

5. Krawetz, Neal. (January/February 2004). IEEE Security and Privacy: *Anti-Honeypot Technology*.

6. Rash, Michael, Orebaugh, Angela, Clark, Graham, Pinkard, Becky & Babbin, Jake (2005). *Intrusion Prevention and Active Response: Deploying Network and Host IPS*. Rockland, MA: Sygress Publishing, Inc.

7. Rowe, Neil C. (January 2006). Proc. 39th Hawaii International Conference on Systems Sciences: *Measuring the Effectiveness of Honeypot Counter-counterdeception*. Poipu, HI.

8. Rowe, Neil C., Duong, B. T., and Custy, John. (June 2006). Proceedings of the 7th IEEE Workshop on Information Assurance: *Fake Honeypots: A Defensive Tactic for Cyberspace*. West Point, NY.

9. SEA-9 (R2C2) Team. (June 2006). Naval Postgraduate School: *Rapid Response Command and Control (R2C2): A System Engineering Analysis of Scalable Communications for Regional Combatant Commanders*.

10. Swiderski, Frank and Synder, Window. (2004). *Threat Modeling*. Redmond, WA: Microsoft Press.

11. Thorsten Holz and Frederic Raynal. (June 2005). Proceedings of the 6th IEEE Workshop on Information Assurance and Security: *Detecting Honeypots and Other Suspicious Environments*. West Point, NY.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Dr. Neil Roweu
   Naval Postgraduate School
   Monterey, California

4. Mr. J. D. Fulp
   Naval Postgraduate School
   Monterey, California

5. Professor Yeo Tat Soon
   Director, Temasek Defence Systems Institute (TDSI)
   National University of Singapore
   Singapore

6. Ms Tan Lai Poh
   Temasek Defence Systems Institute (TDSI)
   National University of Singapore
   Singapore

7. Sze Li Harry, Lim
   Naval Postgraduate School
   Monterey, California

8. Han Chong, Goh
   Naval Postgraduate School
   Monterey, California